International Journal of Computational Intelligence Systems
Vol. **13(1)**, 2020, *pp.* 864–874
DOI: https://doi.org/10.2991/ijcis.d.200617.001; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Research Article

An Integer Cat Swarm Optimization Approach for Energy and Throughput Efficient MPSoC Design

Shahid Ali Murtza[1], Ayaz Ahmad[2], Muhammad Yasir Qadri[3,*], Nadia N. Qadri[2], Majed Alhaisoni[4], Sajid Baloch[5]

[1] *HITEC University, Taxila, Pakistan*
[2] *Department of Electrical & Computer Engineering, COMSATS University Islamabad, Wah Campus, Wah Cantt., Pakistan*
[3] *University of Essex, Colchester, United Kingdom*
[4] *University of Ha'il, Ha'il, Kingdom of Saudi Arabia*
[5] *Institute of System Level Integration (ISLI), University of Edinburgh, Edinburgh, United Kingdom*

ARTICLE INFO

Article History
Received 28 Jul 2019
Accepted 07 May 2020

Keywords
Design space exploration (DSE)
Multicore architecture
Integer cat swarm optimization

ABSTRACT

Modern multicore architectures have an ability to allocate optimum system resources for a specific application to have improved energy and throughput balance. The system resources can be optimized automatically by using optimization algorithms. State-of-the-art using optimization algorithm in the field of such architectures has shown promising results in terms of minimized energy consumption through configuration of number of CPU cores, limited cache sizes and operating frequency. We propose, in this work, a Cat Swarm Optimization (CSO) algorithm-based technique, Integer CSO (ICSO) for the design space exploration (DSE) of multicore computer architectures to find improved energy and throughput balance. The proposed integer variant of CSO algorithm demonstrates convergent behavior for all of design space parameters variations. The Pareto front proposed by ICSO is explored by using various SPLASH-2 benchmarks. Results show significant decrease in energy consumption without affecting throughput severely. Simulation results also validate the use of ICSO in DSE for multicore architectures.

1. INTRODUCTION

Most of the recent research in multicore architectures now focuses on optimum resource allocation according to the workload. This kind of architectures provide better application-specific performance/power ratio due to their ability to adapt as per application requirements. However at design time a designer has to explore a number of configurable parameters in order to strike a balance between energy and throughput of the system. This process is termed as design space exploration (DSE).

Many researchers have used different set of system parameters to tune against different figure of merits in this area of research. Commonly reported parameters used in literature are number of cores, frequency, cache sizes, L_2 cache area, issue width, memory bandwidth, cache associativity, etc. [1–5]. In modern multicore processors, the number of cores integrated can be up to tens, hundreds or even thousands (i.e. in case of GPUs) [6]. Similarly, configurations of cache hierarchy in a multicore architecture system can reach up to numerous possibilities [5]. Handling such a large number of parameters has been an issue for the designers. The higher the number of parameters, the larger the number of configurations in the design space and greater the time needed to find the optimal set of solutions/configurations. Also, with the increase in number of parameters, we need to evaluate more number of system configurations which make a large search/design space which is another challenge. Consequently, simulation of all possible configurations may not always be possible both in terms of time and computational resources [7]. Therefore, two main problems regarding design and analysis phase arise: (1) the huge search space and (2) an increased simulation time. Therefore there exists a need of some optimization algorithm to automatically explore a large design space and propose the system architect a small set of optimal configurations to be simulated or tested.

The size of problem search space can be reduced in some cases when there is a possibility to reduce the number of parameters by eliminating the less important parameters. But by reducing number of parameters, there is a chance to compromise on configurability of the system. On simulation side, we can reduce simulation time to some extent by different techniques of simulations i.e. statistical, sampling and parallel simulation [8]. Even if we reduce the simulation time by different techniques, evaluating all the possible configurations is an NP-hard problem i.e. a lot of time is required to investigate all. Therefore, exhaustive simulation is not possible in most cases. Hence there exists a need of some optimization algorithm to automatically explore a large design space and propose the designer a small set of optimal configurations to be simulated or tested.

[*] *Corresponding author. Email: yasirqadri@acm.org*

Many researchers have used different approaches to obtain improved energy/throughput performance. A number of multicore systems support run-time configuration of voltage and frequency that is termed as Dynamic Voltage/Frequency Scaling (DVFS). To efficiently manage the temperature and energy consumption, various task scheduling, task migration and core consolidation policies can be applied using DVFS. A large body of research is available on DVFS but frequency scaling is not so efficient in many cases as with this technique large unused portions of the chip still dissipate leakage power. In this work, we use the concept of cache re-sizing and core switching along with frequency scaling for better optimization of energy and throughput.

For multicore processors, researcher must consider to best fit the trade-offs in energy consumption, throughput and area which makes it a multiobjective optimization (MOO) problem. The system parameter values in the design space are in integers which renders the problem as a combinatorial optimization problem. Therefore, we need some integer optimization techniques to explore the design space of such type of problems. Therefore, overall the problem becomes Multiobjective Integer Optimization Problem (MIOP). For the DSE in this field, Bio-inspired Algorithms are an important paradigm for consideration. These algorithms are very commonly used to find best and optimum solutions of complex problem in short time. Such algorithms present a shortest way to solve large and time sensitive complex design spaces. Researchers have used many extensions of Bio-inspired Algorithms in the area of DSE e.g. Ant Colony Optimization (ACO) [9,10], Particle Swarm Optimization (PSO) [3], etc.

In this paper, we propose Integer Cat Swarm Optimization (ICSO) algorithm for DSE of a multicore processor architecture. To the best of our knowledge, CSO has not been used with integer optimization specifically in the area of computer design and architecture. We have used the Integer Multiobjective CSO due to its good performance. It has been shown in the literature that the continuous as well as the integer version of CSO has superior performance compared to the well-known and widely used meta-heuristic algorithms such as multiobjective particle swarm optimization (MOPSO) and an improved version of nondominated sorting genetic algorithm (NSGA-II) [11,12]. More specifically, the superior performance of Continuous Multiobjective CSO Algorithm is reported in [12 Section 6.3], whereas the benefits of Integer Multiobjective CSO Algorithm can be witnessed from the results in [11]. The superior performance of the proposed Integer Multiobjective CSO is also clear from the results obtained in this paper. The main difference between the CSO and ICSO is that the parameters in ICSO take integer values and the updating of cats for next iteration is subject to some probability, which makes the ICSO suitable for the DSE problem in hand. In this paper, the ICSO algorithm is intended to support the dynamic configuration of a multicore platform as per work load requirements while maximizing the throughput and minimizing the energy. The algorithm takes number of cores, operating frequency and caches size as design space and searches for an optimal balance between throughput and energy consumption as objective. The algorithm generates a Pareto front between the two objectives i.e. throughput and energy consumption. The best configurations from the generated Pareto front set are evaluated for various benchmark applications using a cycle accurate simulator i.e. Micro Architectural and System Simulator for x86 (MARSSx86) [13]. Results obtained using MARSSx86 show significant reduction in energy consumption without a much impact over throughput.

The remaining paper is organized as follows: Related research in the field of Multicore Processor Architectures is discussed in Section 2. The design space representation is described in Section 3, followed by the implementation of ICSO on the given DSE problem in Section 4. Sections 5 describes the simulation setup and results are provided in Section 6. Finally, conclusion is presented in Section 7.

2. RELATED WORK

Recent research has produced various DSE techniques in the field of Multiprocessor System on Chip (MPSoC) [14,15]. In the following, we provide an account of some of the state-of-the-art approaches in this area.

Calborean *et al.* [6], proposed a framework called Framework for Automatic Design Space Exploration (FADSE). This tool solves single and MOO problems but is unable to handle large design spaces. Mariani *et al.* [1] presented a DSE scheme and an operating system (OS) layer for resource management which targets hardware and software configuration on a multicore platform. However, in this work, the authors took only two configurable parameters i.e. the number of cores and their operating frequency for a particular application. Moreover, their design space comprised of 8-cores whereas our proposed platform is generic because it can solve for any number of cores. Monchiero *et al.* [2], performed DSE for multicore architectures for parameters such as number of cores, L_2 cache size and varying core complexity, for parallel benchmarks to achieve a set of optimal energy/performance trade-offs. We have also considered operating frequency in addition to the parameters taken in their work [2], as a design parameter. We employed swarm intelligence algorithm to explore the given MPSoC's design space.

Givargis *et al.* [16], presented an optimization tool called Platune, which uses parameter independence for characterizing approximate Pareto sets without performing the exhaustive search over the complete design space. Their proposed approach require the specification of parameter independence through a dependency graph. Moreover, the framework supports the exploration of a MIPS based system only.

Researchers have also proposed evolutionary and bio-inspired based algorithms for exploring the design space of MPSoCs [3,4,10,17]. Palermo *et al.* [3], proposed a Discrete Particle Swarm Optimization (DPSO) scheme to perform DSE for hardware of a computing platform. The suggested scheme is aimed to efficiently analyze the workload and to produce an approximated Pareto set of system configurations for selected performance metrics. Although author claimed an 80% reduction in exploration timing as compared to exhaustive search. The effectiveness of the proposed methodology greatly depend on the large population size [18]. Though the efficiency of the algorithm increases with large population size but simulation time still remains a major challenge for complex applications. The exploration scheme considers only caches and CPU as their targeted design space also it lack validation of the proposed solution on any industrial benchmarks. Sheikh *et al.* [4], proposed combined optimization of performance, energy, and temperature i.e. termed as PET optimization. To find optimal solutions, the authors employed multiobjective evolutionary algorithm

(MOEA) and Strength Pareto Evolutionary Algorithm (SPEA). The authors explored number of cores and switching frequency, however, and did not consider the cache size.

For multiprocessor platforms, Beltrame *et al.* [18], proposed design-time analysis strategies that provide multiple mappings for a target application. The authors optimized the energy and delay by decreasing the number of simulations required to find the mappings which provide optimal energy/delay trade-offs. Their proposed scheme takes only two parameters i.e. number of processor cores and size of cache. Singh *et al.* [19], presented a comprehensive survey on mapping methodologies focusing multicore systems. In their paper, the authors discussed various run-time and design-time mapping methodologies along with their advantages and disadvantages in details. Gordon-Ross *et al.* [5], performed DSE on system with a unified L_2 cache as the only configurable parameter.

- Firstly, the DSE of multicore reconfigurable architecture is an integer multiobjective problem, while the algorithm proposed in the literature are either presented for single objective with a constraint or they are not suited for integer multiobjective problem as demonstrated in [11].
- Secondly, we have also compared the proposed methodology with the previously presented nondominated Sorting Genetic algorithm-based DSE methodology and it is being shown that our proposed methodology dominates the NSGA-2 based method.
- Lastly, we have use cycle accurate simulator, the MARSSx86 Splach-2 benchmark applications for the evaluation of the methodology. The obtained results show a significant reduction in energy consumption without a major impact on throughput.

3. REPRESENTATION OF DESIGN SPACE AND MULTIOBJECTIVE PROBLEM

As the complexity of SoC is increasing for multicore processors, one must consider a wide range of issues to best fit the trade-offs in energy consumption, throughput and area. This process is known as DSE involving a MOO problem. Theoretically, multicore architectures can contain many configurable parameters which makes the design space for the search space algorithm very large [5,20,21]. Our design space consists of those configurable system parameters which greatly affect the overall energy consumption and throughput. Our design space includes the cache size, number of cores and operating frequency as described below:

- We take L_1 cache size which can fit into the form of 2^k KB where $k = 0, 1, 2, \cdots, n$. For example, for n = 5, the set of L_1 cache sizes becomes $\{1, 2, 4, 8, 16, 32\}$ KB.
- Number of cores can be any integer ranging from 1 to N where N is the highest number of cores with which the architecture is considered to be equipped.
- The set of operating frequencies of the MPSoC can be represented as $F = f_1, f_2, \cdots, f_m$ where m belongs to a set of positive integer numbers.

Our design space is purely integer where some of the decision variables do not take all integer values between their corresponding lower and upper limits. For the search space algorithm, we use integer index mapping from index based search space to actual search/design space. Here, actual search/design space consists of actual values of the system parameters while index-based search space consists of the corresponding indexes of the set of values in actual design space. In other words, the search space algorithm takes integer indexes of corresponding values in the design space. The corresponding indexes are internally mapped and demapped to actual values in the objective function, as shown in the following example.

For example, if we take our design space as number of cores, 1 to 16, cache sizes $\{1, 2, 4, 8, 16, 32, 128, 256\}$ KB and frequencies $\{10, 16, 25, 33\}$ MHz, then our mapping will be as following:
Actual Design Space $\rightarrow$ Index based Search Space Mapping:
CacheSizes : $\{1, 2, 4, 8, 16, 32, 128, 256\}KB \rightarrow \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
OperatingFrequency : $\{10, 16, 25, 33\}MHz \rightarrow \{1, 2, 3, 4\}$
NumberofCores : $\{1, 2, 3, 4, 5, 6, ..., 16\} \rightarrow \{1, 2, 3, 4, 5, 6, ..., 16\}$ In fact, the considered DSE problem is a combinatorial MOO problem with two conflicting objectives and discrete decision variables (number of cores, cache sizes and frequencies). For multiobjective optimization problem, no single solution exists that simultaneously optimizes each objective and as the two objectives are conflicting, there exists a large number of possible solutions. Even for problems with finite search space, finding the set of Pareto optimal solutions by exhaustive search (complete exploration) has prohibitively high computational complexity. The proposed ICSO with finds the Pareto optimal solution with very small computational complexity. The proposed framework is generalized and can be applied to any size of search space. In the simulations, the range of parameters (search space) is kept small without loss of generality. This is done in order to be able to simulate the framework on ordinary Lab computers as the solution obtained by the hybrid GA-EDA are then evaluated on MARSSx86 simulator and simulations on MARSSx86 simulator are highly time consuming. In future multicore systems, the number of cores, cache sizes and frequencies will be in thousands/millions [22,23] and exhaustive search for design pace exploration in such system would be impossible. For such multicore systems with huge solution space, our proposed DSE framework is equally applicable.

3.1. Multiobjective Problem Representation

The objective we have chosen against these configurable parameters, is to have an optimal balance between energy consumption and throughput of the proposed architecture. The objective function being optimized is presented as follows:

$$F(\mathbf{X}) = \begin{cases} \underset{\mathbf{X}}{\text{minimize}} \quad E(\mathbf{X}) \\ \underset{\mathbf{X}}{\text{maximize}} \quad T(\mathbf{X}) \end{cases} \tag{1}$$

$$\text{where} \quad \mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D}\},$$

$$\text{and } Ran(\mathbf{x_d}) = \{\mathbf{x_d} \in Z^+ | \mathbf{x_d^{min}} \leqslant \mathbf{x_d} \leqslant \mathbf{x_d^{max}} \text{ for } 1 \leqslant d \leqslant D\}.$$

In Equation (1), $E(\mathbf{X})$ and $T(\mathbf{X})$ are the objective functions representing energy consumption and throughput respectively. $\mathbf{X}$ represents the corresponding indexes of the configuration being evaluated consisting of design variables i.e. number of cores, cache size and frequency; $Ran(\mathbf{x_d})$ represents the integer range of $\mathbf{x}_d$ (d^{th} dimension of $\mathbf{X}$) i.e. design variable. Whereas the lower limit is $\mathbf{x_d^{min}}$ and upper limit is $\mathbf{x_d^{max}}$ and D is the total number of design variables i.e. in our case $D = 3$. Note that $\mathbf{X}$ represents indexes of corresponding configuration parameters values which are internally mapped to actual parameter values to achieve actual configuration.

4. CAT SWARM OPTIMIZATION

4.1. History

In 2006, Chu and Tsai [24] introduced a new optimization algorithm i.e. CSO, by modeling the behavior of cats. The authors investigated that cats take rest indolently most of the time when they are awake, move speedily when they are tracing some targets and they are curious about all kinds of moving things. Based upon their investigations, they modeled cats' useful behavior into two modes: Seeking mode (SM) (Resting state) and Tracing mode (TM) (Prey state). Originally, they tested their algorithm for single objective continuous optimization problems. Later on, Pradhan and Panda [25] extended this algorithm to multiobjective continuous optimization problems. Many researchers have also applied CSO in discrete and mixed integer problems involving single objective and multiobjective problems [26–28].

4.2. Proposed Approach

Engineering problems such as MPSoC have their design space in purely integer form and the problem is of combinatorial nature. We propose a new approach, ICSO, to solve this problem. Unlike continuous version of CSO, in proposed ICSO the position vector can take only integer values in each dimension. This approach uses the concept of Asynchronous Updating (AU) proposed for AU-PSO by Zhao *et al.* [29] for mutation. By using this approach, the integer variables in some of the dimensions have a probability to hold the previous values. This kind of treatment provides more chances to find a better solution by better exploring the solution space [29]. These changes make ICSO algorithm different from conventional continuous CSO algorithm. Similar to the continuous version of CSO, proposed ICSO also has two modes: SM and TM.

The ICSO algorithm uses two sets of cats i.e. one set of cats for SM and other for TM to find the optimal solutions. SM gives a global search option while TM provides local search. These two sets of cats jointly solve the optimization problem. For the ratio of cats in the two sets, a parameter called mixture/MR is used which is the ratio of the number of cats in TM to that of the cats in SM. A new parameter ϕ is threshold probability which decides whether to update the new value or keep previous value in a particular dimension of a cat's position.

Each artificial cat has its position, velocity, flag and fitness value. Position of each cat represents a candidate solution and its dimensions indicate the variables of the problem space which are cache size, number of cores (CPU cores) and operating frequency. In other words, position of any cat is the configuration consisting of the combination of L_1 cache, CPU cores and operating frequency. For each iteration, cats are evaluated at their positions, best configurations are stored, and for next iteration cats are modified according to the information based upon best solutions found so far. The objective functions calculate the energy consumption and throughput against the candidate solutions. To this end, energy and throughput models proposed by Qadri *et al.* [30] are used.

The pseudo code of the ICSO is listed in Algorithm 1 and the steps of the algorithm are discussed as following:

4.2.1. Parameters of algorithm

List of general parameters of ICSO algorithm includes swarm size (SwarmSize), dimensions of problem (D), mixing/mixture ratio (MR), maximum iterations to run algorithm (MI) and ϕ. SwarmSize is the total number of cats, whose position represent candidate configurations. D indicates the number of dimensions of the problem i.e. number of input variables of objective function. MR tells how many of the total cats are set for TM and others for SM. The parameter ϕ is threshold probability which decides whether to update the new value or keep previous value in a particular dimension of cat's position. Default values for these parameters are shown in the Table 1.

SM has four special parameters: Seeking Memory Pool (SMP), Self Position Consideration (SPC), Counts of the Dimension to Change (CDC) and Range of the Selected Dimension (SRD). SMP is the number copies to be made of current configuration and SPC is a flag that indicates whether current configuration will be one of the configurations to be selected again or not. CDC indicates how many of the inputs from number of cores, cache size and frequency are to be varied. SRD defines the mutation ratio for the selected input.

TM has two parameters c_1 and r_1. Here, c_1 is the algorithm's cognitive learning constant which can have a value from 1 to 4 and r_1 is a random number taken from 0 to 1.

The pseudo code of the ICSO is shown in Algorithm 1 and the detail, and steps of the algorithm are as following:

4.3. Algorithm

Following are the main steps in the ICSO algorithm.

Step 1. Initialize the algorithm parameters and Cats positions.

The algorithm parameters of ICSO are specified in first step i.e. the number of initial solutions (SwarmSize), MR, maximum number of iterations to run the algorithm (MI), asynchronous probability (ϕ) and the Seeking and TM parameters. This step of ICSO algorithm also involves the initialization of variables. In this step, the selection of initial configurations comprising of random values of cache size, number of cores and frequency is done from the specified design space. Each artificial cat at its position represents a candidate solution/configuration where each dimension of the position of cat is a design variable. In Algorithm 1, lines 1 to 5 represent this step.

Table 1 Parameters of ICSO algorithm.

Swarmsize	MR	SMR	SRD	CDC	$\mathbf{V_{max}}$
50	0.3	3	0.2	0.2	4

ICSO, Integer Cat Swarm Optimization; CDC, Counts of the Dimension to Change; SRD, Range of the Selected Dimension.

Algorithm 1 ICSO: Pseudo Code of Integer Cat Swarm Optimization

```
1: Inputs: $NumberofCores$, $OperatingFrequency$, $CacheSizes$
2: Initialize Parameters:
   $\{SwarmSize, D, MR, MI, SMP, SPC, CDC, SRD, v_{max}, c_1, \phi\}$
3: $Cats \leftarrow \{\ \}$
4: for $SwarmSize$ times do                                ▷ Cats initialization
5:     $Cats \leftarrow Cats \cup$ {new $k^{th}$ cat with position $X_k$, velocity $V_k$, flag $F_k$ and fitness $FS_k$}
6: $Archive \leftarrow ParetoFrontUsingParetoDominance(FS)$
7: repeat
8:     for each cat $X_k \in Cats$ do
9:         if flag of cat $X_k$ is true then                ▷ Cat is in Tracing mode
10:            $X_g \leftarrow$ one of the optimal cats, found so far.
11:            $r_1 \leftarrow U(0,1)$              ▷ $U(0,1)$ is a random number from [0,1]
12:            for each dimension $d$ do
13:                $r_{kd} \leftarrow U(0,1)$
14:                $v'_{kd} \leftarrow v_{kd} + c_1 * r_1 * (x_{gd} - x_{kd})$
15:                if $v'_{kd} > v_{max}$ then $v'_{kd} \leftarrow v_{max}$
16:                if $round(x_{kd} + v'_{kd}) \in Range(x_d)$ and $r_{kd} < \phi$ then
17:                    $x_{kd} \leftarrow round(x_{kd} + v'_{kd})$
18:                else
19:                    $x_{kd} \leftarrow x_{kd}$
20:            $FS \leftarrow ObjectiveFunction(X_k)$.
21:            $Archive \leftarrow ParetoFrontUsingParetoDominance(Archive \cup FS)$.
22:        else                                           ▷ Cat is in Seeking Mode
23:            $CatsCopies \leftarrow \{j = SMP - SPC$ copies of cat $X_k\}$
24:            for each copy $Y_l \in CatsCopies$ do
25:                for each dimension $d$ do
26:                    $r_{ld} \leftarrow U(0,1)$
27:                    $y_{ld} \leftarrow y_{ld} \pm SRD * r_{ld}$ with $CDC$ taking into account
28:                    if $round(y_{ld}) \in Range(y_d)$ and $r_{ld} < \phi$ then
29:                        $y_{ld} \leftarrow round(x_{ld})$
30:                    else
31:                        $y_{ld} \leftarrow x_{kd}$
32:            $FS \leftarrow ObjectiveFunction(Y)$.          ▷ $Y$: All mutated copies.
33:            $Archive \leftarrow ParetoFrontUsingParetoDominance(Archive \cup FS)$.
34:            $X_k \leftarrow SelectCopyRandomlyFrom(Y)$
35:    Set cats randomly into TM (flag:true) & SM (flag:false) according to $MR$.
36: until algorithm meets termination criteria.
37: return $Archive$
```

Step 2. Initialize the velocity and flag.

In Algorithm 1, lines 4 to 5 represent the second step. This step involves the initial settings of algorithm mechanism variables i.e. velocity and flags. Movements of cats are modeled by their velocities and their positions are defied based upon the values of velocities. Initially, random velocities (V) are assigned within a given range to each dimension. To ensure the velocities to be within a given range during algorithm run, a threshold value, v_{max} needs to be specified according to velocity ranges. Since positions and movements of cats are grouped in two modes, Tracing and Seeking, therefore mixture/mixing ratio (MR) parameter is used to specify how many cats will go either into TM or SM. Each cat is set a flag representing its mode. By default, all cats are in SM and randomly chosen cats are set into TM according to MR.

Step 3. Evaluate objective function and find Pareto front.

In Algorithm 1, line 6 represents this step. Initially created cats are evaluated for the objective function at their position (X) as a candidate solution. The results of the objective functions are evaluated to find the Pareto optimal set from objective values using Pareto dominance concept given in [25] and the result is stored in an external archive.

Step 4. Applying cats into corresponding mode.

After the initial/latest Pareto front is obtained, the cats are iterated through a loop and based upon their flag they are set into SM process (Section 4.3.2) and TM process (Section 4.3.1). In Algorithm 1, lines 9 to 21 represent TM and lines 22 to 34 represent SM. The overall step is implemented from line number 8 to 34.

Step 5. Repeat step 4 and so on, until the termination criterion is met.

In the last step, the termination criteria that may be specific number of iterations completed or ideal solution is examined, if it is satisfied *Archive* is returned as an output and the program is terminated. Otherwise, cats are remixed into TM and SM randomly according to MR and the program control is jumped back to step 4 for newly evaluated cats. This step is shown in Algorithm 1 in lines 35 to 37.

4.3.1. Tracing mode

In Algorithm 1, lines 9 to 21 represent the TM. This mode gives a local search option in exploring the design/search space. In this mode, a cat chases the target with high speeds which is mathematically modeled as large changes in the cat's position i.e. cat takes larger steps towards convergence. In the D-dimensional space, the position of the k^{th} cat is represented as $X_k = (x_{k1}, x_{k2}, \cdots, x_{kD})$ and its velocity is represented as $V_k = (v_{k1}, v_{k2}, \cdots, v_{kD})$, where the second subscripts $d = \{1, 2, \cdots, D\}$ represent the dimension number. The global best position of the cat swarm is presented as $X_g = (x_{g1}, x_{g2}, \cdots, x_{gD})$. The steps involved in this mode are as follows:

1. Update the velocity of cat_k by the following equation:

$$v'_{kd} = v_{kd} + c_1 * r_1 * (x_{gd} - x_{kd}) \tag{2}$$

   where x_{kd} and v_{kd} are position and velocity of cat_k in dimension d respectively and x_{gd} is the position of best cat in that dimension found so far. Here, c_1 is the algorithm constant which can have a value from 1 to 4 and r_1 is a randomly generated number between 0 to 1 inclusive. The global best X_g is taken randomly from the external archive, that is generally picked from the top five percent of the previously obtained nondominated solutions.

2. Check velocity for v_{max} and clamp if $v_{kd} > v_{max}$.
3. Update the position of cat_k by the following equation:

$$x_{kd} = \begin{cases} round(x_{kd} + v'_{kd}) \\ \text{if } round(x_{kd} + v'_{kd}) \in Ran(x_d) \text{ and } r_{kd} < \phi \\ x_{kd} \\ \text{otherwise} \end{cases} \tag{3}$$

   where r_{kd} is a random value from $[0, 1]$, $Ran(x_d)$ is the range of x_d and ϕ is a threshold value which dictates AU. Also $d = \{1, 2, \cdots, D\}$.

4. Evaluate objective function at X_k.
5. Update *Archive* if current solution is nondominated.

4.3.2. Seeking mode

In Algorithm 1, lines 22 to 34 represent the SM also known as resting mode of cats. This step in proposed integer variant of CSO differs from conventional CSO in updating technique (refer Section 4.2) for positions of cats. The steps involved in this mode are given below:

1. Make $j = SMP - SPC$ copies of cat_k. (Each copy is represented by $y : y_{ld}$, where $1 < l \leqslant j$ and $1 < d \leqslant D$, l is cat's copy number and d is the dimension number.)
2. Apply mutation operator to each copy.
3. Check for bounds of each dimension of all copies and update using the following equation.

$$y_{ld} \leftarrow \begin{cases} round(y_{ld}) & \text{if } round(y'_{ld}) \in Ran(x_d) \text{ and } r_{ld} < \phi \\ x_{kd} & \text{otherwise} \end{cases} \tag{4}$$

   where r_{kd} is a random real value from $[0, 1]$, $Ran(x_d)$ represents range of x_d and ϕ is a threshold value which dictates AU.

4. Evaluate objective function at all mutated copies.
5. Update *Archive* with those modified copies that give nondominated solutions/configurations.
6. Select a candidate randomly from j copies which replaces the older position (X_k) of cat_k.

5. SIMULATION SETUP

This section discusses the multicore configurable processor architecture and the simulation setup that is used to evaluate the proposed scheme. Our multicore architecture platform takes L_1 cache sizes, number of CPU cores and operating frequency as configurable variables. The objective of our proposed work is to have improved energy and throughput balance i.e. minimizing energy consumption with little compromise on throughput. To meet this objective, we can vary the values of decision variables within a specific range to get different configurations.We vary number of symmetric cores from 1 to 16. The platform incorporates configurable L_1 cache size, number of cores and operating frequency/-voltage. We used ICSO algorithm that takes frequency, number of cores and caches sizes as design space and finds an efficient balance between energy consumption and throughput as an objective. This algorithm was coded in MATLAB 8.1 (R2013a). This algorithm gives a Pareto front set of efficient balances between energy consumption and throughput. Then, we used a full system simulator MARSSx86 [13] to simulate the optimal solutions suggested by ICSO. This simulator is a fast cycle accurate simulation platform for single core and multicore configurations. It is targeted to simulate cycle accurate out-of-order x86 cores. In a single simulation run, it gives statistics of user and kernel modes activities separately. We defined an x86 based 16-core platform in which each core has private L_1 cache and all cores are coupled with a single shared L_2 cache. We used Modified-Shared-Exclusive-Invalid (MESI) protocol [36] for the coherence between cache and main memory. The experiments were performed on Dell OptiPlex core i5 system with 8Gb of ram and 32Ghz processor frequency. We used Ubuntu version 10.04 as the targeted OS for simulation purpose. We obtained L_1 cache timing and energy information from CACTI [37]. However, CACTI simulator is not a trace driven, therefore it does not account energy consumption due to number of cache hits and misses for a specific application. Therefore, we used detailed analytical models presented by Qadri *et al.* [30] to estimate the cache throughput and energy based on cache hit/miss information.

Table 2 Benchmark applications used in this work.

Sr.	Application	Description	Problem Size
1	Barnes	Employs Barnes-Hut algorithm to solve N-body problems [31].	16,384 particles
2	FMM	Uses a parallel adaptive Fast Multipole Method (FMM) to simulate the N-body problems [32].	16,384 particles
3	Ocean	Simulate large-scale ocean motility based on eddy currents and boundary currents [33].	258x258 grid
4	Water-NSquared	An improvised version of the original Water code in SPLASH [34,35].	512 molecules
5	Water-Spatial	Simulates the same molecular dynamics N-body problem as in the original Water-NSquared code in SPLASH [34].	512 molecules

Table 3 Pareto front (ICSO).

Cache sizes(KB)	No. of cores	Frequency (MHz)	Throughput	Energy(J)
2	1	10	1.02986	0.05486
2	1	25	2.57466	0.05487
4	1	33	3.39006	0.08506
2	2	33	3.99855	0.10973
1	3	33	4.40376	0.11688
1	5	33	4.90376	0.19480
1	6	33	5.07043	0.23376
1	7	33	5.20376	0.27272
1	8	33	5.31285	0.31168
1	9	33	5.40376	0.35064
1	10	33	5.48069	0.38960
1	11	33	5.54662	0.42856
1	12	33	5.60376	0.46752
1	13	33	5.65376	0.50648
1	14	33	5.69788	0.54544
1	15	33	5.73710	0.58440
1	16	33	5.77218	0.62336
32	16	33	5.78840	5.26334
64	15	33	5.79334	5.61560
64	16	33	5.82842	5.98998

ICSO, Integer Cat Swarm Optimization.

We used Intel 486 GX embedded processor data sheet [38] for energy consumption calculations and frequency range. We evaluated the optimal solution, suggested by ICSO, using SPLASH-2 benchmark applications. All the applications were executed for the complete run and then design space parameters were modified based on results obtained using ICSO algorithm.

As the suggested architecture consists of a multicore platform, a set of SPLASH-2 [39] benchmark applications is chosen in order to perform the objective evaluation. A summary of the benchmark applications used for this purpose is given in Table 2.

6. RESULTS

6.1. Algorithm Results

As the main objective of the proposed approach is to achieve a balance between throughput and energy consumption of the SoC. In order to achieve this, the ICSO algorithm-based technique takes the configurations consisting of system parameters i.e. number of cores, L_1 cache size and operating frequency (for testing purpose, we take the limited ranges of these parameters; the ranges of the parameters are as given in the example in Section 3). This algorithm is run on 100 iterations with 20 number of initial solutions, and a 0.5 value of asynchronous probability (ϕ). The obtained Pareto front is shown in Figure 1 and Table 3. This Pareto front has 20 optimal configurations. Figure 1 shows that initially throughput increases with the increase in energy consumption and then throughput almost saturates but energy consumption goes on to increase. If we look on Table 3, we can see the corresponding configurations for throughput and energy consumption shown in Figure 1. For comparison, we also obtained the true Pareto front for the same ranges of configuration parameters and a Pareto front using the integer variant of NSGA-II [40] and EDA algorithm [14]. A comparison of these four Pareto fronts (true Pareto front and Pareto fronts obtained from NSGA-II, Pareto front of EDA and proposed ICSO algorithm) are shown in Figure 1. The plot clearly shows that the Pareto front obtained from the proposed ICSO algorithm approximately overlaps on true Pareto front while the NSGA-II and EDA Pareto fronts are much deviant from true Pareto front curve. Hence the proposed ICSO algorithm proves to be useful for integer type of problems with better results.

6.2. Evaluation of Pareto Set

To evaluate the Pareto set obtained using ICSO algorithm, we have selected two configurations from the set of twenty configurations shown in Table 3 for detailed simulations i.e. one configuration with minimum energy consumption, we call it energy efficient configuration and second configuration with maximum throughput, we call it throughput efficient configuration. Then, we have a default system configuration, which uses maximum available system resources, as a reference to compare the effect of system parameter variations in energy efficient and throughput efficient configurations. The default, throughput efficient and energy efficient configurations are shown in Table 5. The configurations given in this table are simulated with different SPLASH-2 benchmark applications (Barnes, Cholesky, LU, FMM, Water-Spatial and Water-NSquared) on MARSSx86 simulator. It should be noted that the default system configuration has maximum throughput but at the same time maximum energy consumption. The results of other two configurations, throughput efficient and energy efficient, are discussed with reference to this default configuration as following.

Table 4 Pareto front energy consumption and throughput values for NSGA-2II and True Pareto-front.

					NSGA-II Pareto-front						
Throughput	6.188	5.775	5.389	5.363	5.050	5.046	4.950	4.570	4.554	4.541	4.537
Energy consumption	4.125	3.300	3.045	2.888	0.570	0.568	0.413	0.078	0.026	0.011	0.0108
					EDA Pareto-front						
Energy	0.13	0.47	1.26	2.36	2.52	3.4	4.36	6.24	8.3	10.58	10.84
Throughput	2.41	3.17	5.37	5.42	5.74	5.91	6.06	7.06	7.18	8.84	10.37
					True Pareto-front						
Throughput	5.828	5.793	5.772	5.204	5.070	4.404	4.004	3.404	2.579	1.650	1.031445
Energy consumption	5.990	5.616	0.623	0.273	0.234	0.117	0.078	0.039	0.039	0.039	0.038

NSGA-II, nondominated sorting genetic algorithm.

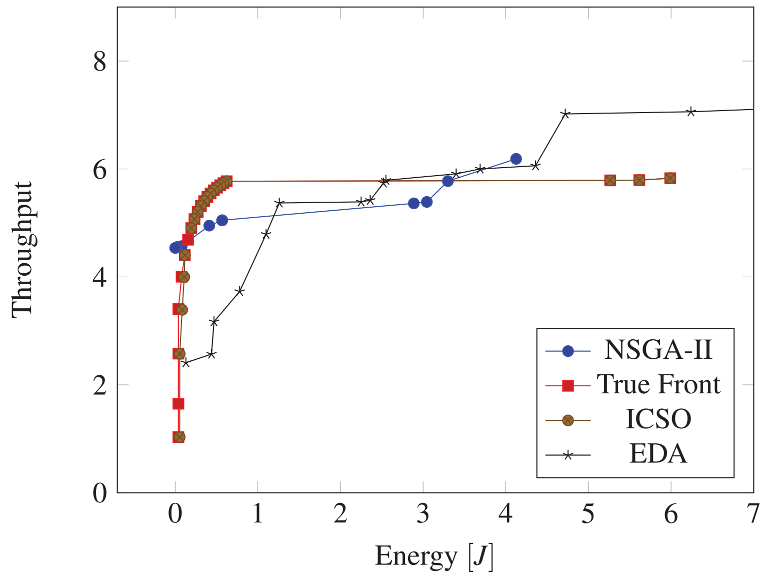

Figure 1 Pareto-front (set of nondominated solutions).

Table 5 Selected configurations for MARSSx86 simulation.

Configuration Name	Cache Size(KB)	Core	Frequency(MHz)
Default system	256	16	33
Throughput efficient	64	16	33
Energy efficient	2	1	10

MARSSx86; Micro Architectural and System Simulator for x86.

A. L_1 Cache

L_1 cache size has a significant impact on throughput and energy consumption of an MPSoC. Therefore it is necessary to include L_1 cache size variations for analysis. For performance measurement of L_1 cache the performance metric miss ratio is used. L_1 cache miss ratio depends on the size of the cache. The larger L_1 cache size results in fewer misses. However larger cache size also increases latency, and thus consumes more power. For simulation, all SPLASH-2 applications are set to L_1 cache size of 256KB, 64KB, and 1KB for default, throughput efficient and energy efficient configuration respectively. Simulation results for L_1 cache miss ratio for all three configurations are shown in Figure 2. The figure shows that reducing cache size from 256KB to 64KB does not effect much on all benchmark applications except for Ocean for which the aggregate L_1 miss ratio increases to 10% from default 5%. For energy efficient configuration reducing L_1 cache size from 256KB to 1KB increases the aggregate L_1 miss ratio from 15% to 20% for FMM, Ocean, and Water-Spatial as compared to the default ranges from 1% to 5%. For Barnes and WWater-Nsquared the miss ratio remains almost the same. The proposed architecture makes the use of a shared level 2 (L_2), uniform cache memory. For simulation, all SPLASH-2 applications are set to L_2 cache size of 2MB for all three configurations. Simulation results for L_2 cache miss ratio for all three configurations are shown in Figure 3. This figure shows that for throughput efficient configuration L_2 miss ratio is reduced significantly except for Ocean and Water-Nsquared for which it is increased from 1% to 5% and 25% to 30% respectively. For energy efficient configuration an overall reduction in miss ratio is observed for almost all the applications. The least reduction in miss ratio can be observed in case of Barnes for which it reduces from 38% to 35% only. A nominal increase of 2% is observed for Ocean benchmark.

B. Number of Cores and CPU Frequency

Number of cores and CPU frequency has a significant impact on throughput and energy consumption. Increase in CPU frequency increases throughput greatly along with increase in energy consumption. Since for increased operating frequency, a system takes less time to complete a task and the overall average energy consumption can be more or less same. While on the other hand, throughput is not increased linearly by increasing the number of cores rather relationship between throughput and number of cores

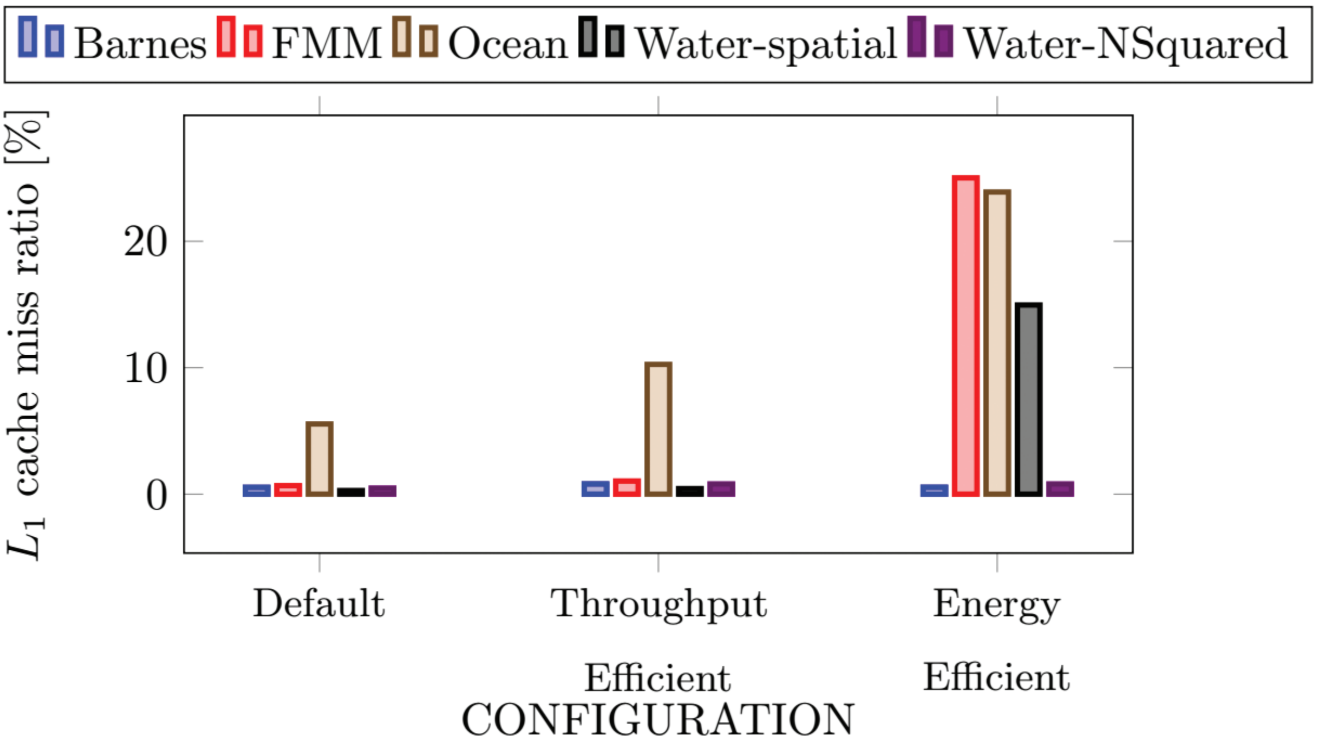

Figure 2 L_1 cache miss ratio.

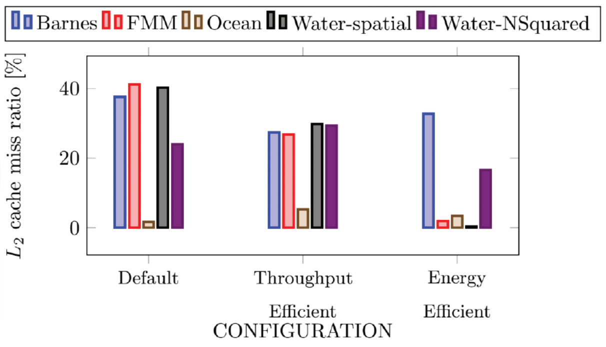

Figure 3 L_2 cache miss ratio.

is governed by Amdhal's Law [41]. Throughput saturates for the most of the applications after scaling the number of cores beyond a certain limit [42]. Energy consumption is also increased with the increase of number of cores, which can be reduced by turning off of excess cores [43]. In selected pareto set, number of cores for default and throughput efficient configurations is 16, and for energy efficient it is 1. The frequency for these configurations is set to 33MHz, 33MHz and 10MHz for default, throughput efficient and energy efficient configuration respectively. These values of frequency and number of core are shown in Table 5, and are suggested by ICSO algorithm.

Results for normalized throughput and energy are shown in Figures 4 and 5 respectively. The figures clearly show that the default configuration has maximum throughput and at the same time maximum energy consumption. Throughput efficient configuration has reduced throughput and it also has reduced energy consumption. The minimum throughput is observed for Ocean benchmark i.e. 75% of the default value (see Figure 4). Whereas for all other applications less than 5% throughput reduction. The throughput efficient configuration shows decrease in energy consumption for all the applications (see Figure 5). Barnes application has shown least energy consumption i.e. around 10% of the default configuration. Whereas Ocean shows energy savings of up to 40% and for all other applications it is around 50%. The energy efficient configuration showed throughput reduction of more than 90% for all applications that is not desirable for most cases however this has resulted in highest energy savings of 90% or more (see Figures 4 and 5). Thus, results show that the throughput efficient configuration has desirable improvements in terms of both throughput and energy consumption.

Figure 6 shows the normalized Energy Delay Product (EDP) of the three configurations. The figure shows that the EDP of throughput efficient configuration is 53% of the default value and of energy efficient configurations 55% of the default value on average (see Figure 6). The minimum EDP is observed for Barnes application for throughput efficient configuration i.e. 13% of the default value. For this configuration, FMM, Water Spatial and Water-NSquared show about 50% reduction in EDP. For energy efficient configuration, Barnes achieves 19% EDP and FMM, Ocean, Water Spatial and Water-NSquared achieve about 65% of the default EDP. Overall EDP is reduced significantly in all applications except Ocean application which improved least with 98% and 87% of the default EDPs for throughput efficient and energy efficient configurations respectively.

Since lower EDP is desired one, thus the results show a significant overall improvement in performance of these configurations suggested by the proposed ICSO algorithm.

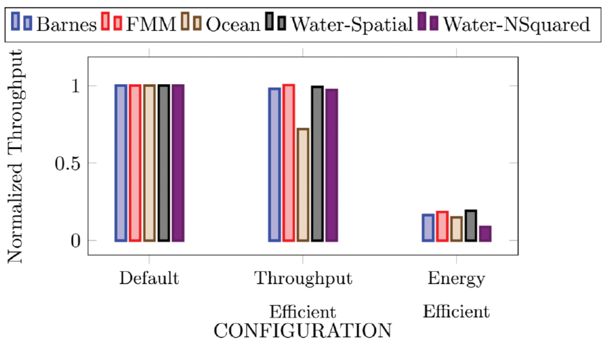

Figure 4 Results for normalized throughput.

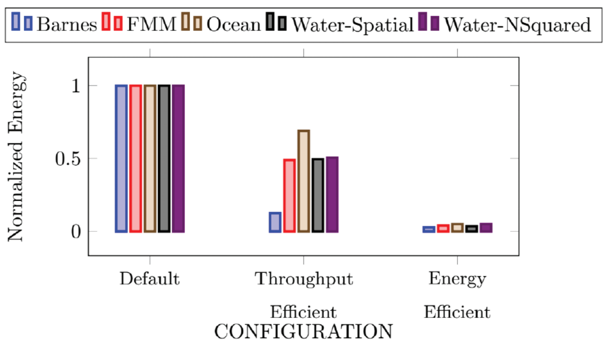

Figure 5 Results for normalized energy.

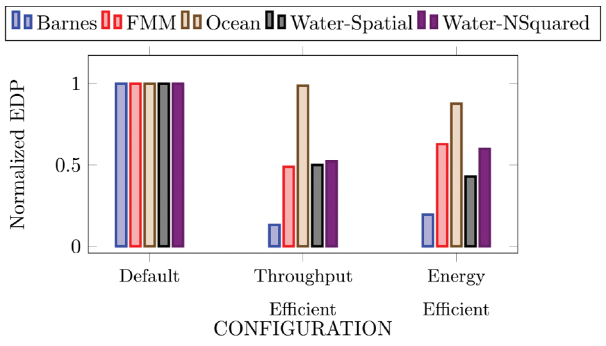

Figure 6 Results for energy delay product.

7. CONCLUSION

In this paper, ICSO-based multiobjective DSE methodology was presented for multicore reconfigurable architectures. The objective of proposed methodology was to find an optimal trade-off between energy consumption and throughput. The pareto curve obtained by the proposed methodology was compared with the other state of the art exploration methodology which demonstrate the dominance of our proposed ICSO-based exploration method. The results were evaluated using extensive simulation which includes cycle accurate simulator MARSSx86, SPLASH-2 benchmark applications and CACTI for L1 cache energy consumption and timing data. The results demonstrate significant reduction in energy consumption without a major impact on the throughput. The proposed ICSO-based exploration technique is demonstrated for MPSoC reconfiguration but it is extendable for many core processors with more design space parameters.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHORS' CONTRIBUTIONS

Shahid Ali Murtza Algorithm development, Writing Original Draft, Simulations, Results presentation; Ayaz Ahmad Algorithm development, Supervision in simulations, Results analysis; Muhammad Yasir Qadri Multicore modelling, target platform parametrization, final result evaluation, Final draft preparation, Project supervision (Project Co-PI), ICT R&D Funding; Nadia N. Qadri Data analysis, Development of Methodology; Majed Alhaisoni Project supervision, data analysis, Manuscript finalization, University of Ha'il Funding; Sajid Baloch Project supervision, team management and administration, validation and verification of results

ACKNOWLEDGMENTS

This work was part supported by the National ICT R&D Fund Pakistan, through grant number: ICTRDF/TR&D/2012/65 and by Deanship of Scientific Research at University of Ha'il, Ha'il, Kingdom of Saudi Arabia.

REFERENCES

[1] G. Mariani,P. Avasare, G. Vanmeerbeeck, C.Y. Couvreur, G. Palermo, C. Silvano, V. Zaccaria, An industrial design space exploration framework for supporting run-time resource management on multi-core systems, in Proceeding of Conference & Exhibition on Design, Automation & Test in Europe (DATE), Dresden, Germany, 2010, pp. 196–201.
[2] M. Monchiero, R. Canal, A. González, Design space exploration for multicore architectures: a power/performance/thermal view, in Proceeding of 20th Annual International Conference on Supercomputing, Cairns, Australia, 2006, pp. 177–186.
[3] G. Palermo, C. Silvano, V. Zaccaria, Discrete particle swarm optimization for multi-objective design space exploration, in Proceeding of 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'08), Parma, Italy, 2008, pp. 641–644.
[4] H.F. Sheikh, I. Ahmad, Simultaneous optimization of performance, energy and temperature for DAG scheduling in multicore processors, in International Green Computing Conference (IGCC), San Jose, CA, USA, 2012, pp. 1–6.
[5] A.G. Ross, F. Vahid, N. Dutt, Fast configurable-cache tuning with a unified second-level cache, IEEE Trans. Very Large Scale Integration Syst. 17 (2009), 80–91.
[6] H. Calborean, L. Vintan, An automatic design space exploration framework for multicore architecture optimizations, in International Conference on Roedunet (RoEduNet), Sibiu, Romania, 2010, pp. 202–207.
[7] L. Vintan, Main challenges in multicore architecture research, Revista Romana de Informatica si Automatica. 19 (2009).
[8] J.J. Yi, D.J. Lilja, Simulation of computer architectures: simulators, benchmarks, methodologies, and recommendations, IEEE Trans. Comput. 55 (2006), 268–280.
[9] E. Kugu, O.K. Sahingoz, ACO algorithms with multi-core implementation, in Proceeding of 7th InternationalConference on

Application of Information and Communication Technologies (AICT), Baku, Azerbaijan, 2013, pp. 1–5.
[10] I. Hussain, A. Ahmad, M.Y. Qadri, N.N. Qadri, J. Ahmed, Ant colony optimization for multicore re-configurable architecture, AI Commun. 29 (2016), 595–606.
[11] S.A. Murtza, A. Ahmad, J. Shafique, Integer cat swarm optimization algorithm for multiobjective integer problems, Soft Comput. 24 (2020), 1927–1955.
[12] B. Xing, W.-J. Gao, Innovative Computational Intelligence: a Rough Guide to 134 Clever Algorithms, Intelligent Systems Reference Library, vol. 62, Springer, Cham, Switzerland, 2014.
[13] A. Patel, F. Afram, K. Ghose, MARSS-x86: a QEMU-based microarchitectural and systems simulator for x86 multicore processors, in Proceeding of 1st International QEMU Users Forum, Grenoble, France, 2011, pp. 29–30.
[14] M. Murad, I. Hussain, A. Ahmad, M.Y. Qadri, N.N. Qadri, Estimation of distribution based multi-objective design space exploration for energy and throughput-optimized MPSoCs, Turk. J. Electr. Eng. Comp. Sci. 28 (2020), 540–555.
[15] I. Hussain, S.A. Murtza, M.Y. Qadri, M. Fleury, N.N. Qadri, Scalable, energy-aware system modeling and application-specific reconfiguration of MPSocs with a type-2 fuzzy logic system, Comput. Electr. Eng. 74 (2019), 292–304.
[16] T. Givargis, F. Vahid, Platune: a tuning framework for system-on-a-chip platforms, IEEE Trans. Comput. Aided Design Integr. Circuits Syst. 21 (2002), 1317–1327.
[17] I. Hussain, A. Parveen, A. Ahmad, M.Y. Qadri, N.N. Qadri, J. Ahmed, NSGA-II based design space exploration for energy and throughput aware multicore architectures, Cybern. Syst. 48 (2017), 536–550.
[18] G. Beltrame, L. Fossati, D. Sciuto, Decision-theoretic design space exploration of multiprocessor platforms, IEEE Trans. Comput. Aided Design Integr. Circuits Syst. 29 (2010), 1083–1095.
[19] A.K. Singh, M. Shafique, A. Kumar, J. Henkel, Mapping on multi/many-core systems: survey of current and emerging trends, in Proceeding of 50th Annual Design Automation Conference, Austin, TX, USA, 2013, pp. 1–10.
[20] K. Sigdel, System-Level Design Space Exploration of Reconfigurable Architectures, PhD Thesis, Delft University of Technology, TU Delft, 2011.
[21] C. Silvano, W. Fornaciari, E. Villar, Multi-objective Design Space Exploration of Multiprocessor SoC Architectures, Springer, New York, NY, USA, 2011.
[22] J. Dongarra, S. Gottlieb, W.T.C. Kramer, Race to exascale, Comput. Sci. Eng. 21 (2019), 4–5.
[23] S. Furber, A. Brown, Biologically-inspired massively-parallel architectures - computing beyond a million processors, in Proceedingof the Ninth International Conference on Application of Concurrency to System Design, Augsburg, Germany, 2009, pp. 3–12.
[24] C. Shu-Chuan, P. Tsai, Computational intelligence based on the behavior of cats, Int. J. Innov. Comput. Inf. Control. 3 (2007), 163–173.
[25] P.M. Pradhan, G. Panda, Solving multiobjective problems using cat swarm optimization, Expert Syst. Appl. 39 (2012), 2956–2964.
[26] D. Kumar, S.R. Samantaray, I. Kamwa, N.C. Sahoo, Reliability-constrained based optimal placement and sizing of multiple distributed generators in power distribution network using cat swarm optimization, Electr. Power Compo. Syst. 42 (2014), 149–164.
[27] B. Santosa, M.K. Ningrum, Cat swarm optimization for clustering, in Proceeding of Soft Computing and Pattern Recognition (SOCPAR'09), Malacca, Malaysia, 2009, pp. 54–59.
[28] Y. Sharafi, M.A. Khanesar, M. Teshnehlab, Discrete binary cat swarm optimization algorithm, in Proceeding of 3rd International Conference on Computer, Control & Communication (IC4), Karachi, Pakistan, 2013, pp. 1–6.
[29] X. Zhao, Y. Jin, H. Ji, J. Geng, X. Liang, R. Jin, An improved mixed-integer multi-objective particle swarm optimization and its application in antenna array design, in Proceeding of 2013 IEEE 5th International Symposium Microwave, Antenna, Propagation and EMC Technologies for Wireless Communication (MAPE), Chengdu, China, 2013, pp. 412–415.
[30] M.Y. Qadri, K.D. McDonald-Maier, Analytical evaluation of energy and throughput for multilevel caches, in Proceeding of 12th International Conference on Computer Modelling and Simulation (UKSim), Cambridge, UK, 2010, pp. 598–603.
[31] J.P. Singh, J.L. Hennessy, A. Gupta, Implications of hierarchical n-body methods for multiprocessor architectures, ACM Trans. Comput. Syst. 13 (1995), 141–202.
[32] J.P. Singh, C. Holt, J.L. Hennessy, A. Gupta, A parallel adaptive fast multipole method, in Proceeding of 1993 ACM/IEEE Conference on Supercomputing, Portland, OR, USA, 1993, pp. 54–65.
[33] S.C. Woo, J.P. Singh, J.L. Hennessy, The Performance Advantages of Integrating Message Passing in Cache-Coherent Multiprocessors, Technical Report, Stanford University, 408 Panama Mall, Suite 217, Stanford, CA, United States, 1993.
[34] C.W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, NJ, USA, 1971.
[35] J.P. Singh, W.-D. Weber, A. Gupta, SPLASH: stanford parallel applications for shared-memory, ACM SIGARCH Comp. Arch. News. 20 (1992), 5–44.
[36] M.S. Papamarcos, J.H. Patel, A low-overhead coherence solution for multiprocessors with private cache memories, ACM SIGARCH Comp. Arch. News. 12 (1984), 348–354.
[37] D. Tarjan, S. Thoziyoor, N.P. Jouppi, CACTI 4.0, Technical Report, HPL-2006-86, HP Laboratories, Palo Alto, CA, USA, 2006.
[38] Intel, Embedded Ultra-Low Power Intel-486 GX Processor, Datasheet, Intel Corporation, Santa Clara, CA 95054-1549, USA, 1997, pp. 47–48.
[39] S.C. Woo, S. Cameron, M. Ohara, E. Torrie, J.P. Singh, A. Gupta, The SPLASH-2 programs: characterization and methodological considerations, ACM SIGARCH Comp. Arch. News. 23 (1995), 24–36.
[40] R.F. Subtil, E.G. Carrano, M. Souza, R.H.C. Takahashi, Using an enhanced Integer NSGA-II for solving the multiobjective generalized assignment problem, in Proceeding of 2010 IEEE Congress on Evolutionary Computation (CEC), Barcelona, Spain, 2010, pp. 1–7.
[41] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in Proceeding of the Spring Joint Computer Conference (AFIPS '67), Spring, New York, NY, USA, 1967, pp. 483–485.
[42] J.L. Gustafson, Re-evaluating Amdahl's law, Commun. ACM. 31 (1988), 532–533.
[43] P. Thanarungroj, C. Liu, Power and energy consumption analysis on Intel SCC many-core system, in Proceeding of IEEE 30th International Performance Computing and Communications Conference (IPCCC), Orlando, FL, USA, 2011, pp. 1–2.