Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 6382765, 11 pages
http://dx.doi.org/10.1155/2016/6382765

Research Article

An Efficient Technique for Hardware/Software Partitioning Process in Codesign

Imene Mhadhbi, Slim Ben Othman, and Slim Ben Saoud

Department of Electrical Engineering, National Institute of Applied Sciences and Technology, Polytechnic School of Tunisia, Advanced Systems Laboratory, B.P. 676, 1080 Tunis Cedex, Tunisia

Correspondence should be addressed to Imene Mhadhbi; imene.mhadhbi@gmail.com

Received 27 January 2016; Accepted 10 May 2016

Academic Editor: Michele Risi

Codesign methodology deals with the problem of designing complex embedded systems, where automatic hardware/software partitioning is one key issue. The research efforts in this issue are focused on exploring new automatic partitioning methods which consider only binary or extended partitioning problems. The main contribution of this paper is to propose a hybrid FCMPSO partitioning technique, based on Fuzzy C-Means (FCM) and Particle Swarm Optimization (PSO) algorithms suitable for mapping embedded applications for both binary and multicores target architecture. Our FCMPSO optimization technique has been compared using different graphical models with a large number of instances. Performance analysis reveals that FCMPSO outperforms PSO algorithm as well as the Genetic Algorithm (GA), Simulated Annealing (SA), Ant Colony Optimization (ACO), and FCM standard metaheuristic based techniques and also hybrid solutions including PSO then GA, GA then SA, GA then ACO, ACO then SA, FCM then GA, FCM then SA, and finally ACO followed by FCM.

1. Introduction

The hardware/software partitioning process presents the crucial task of the codesign methodology. It is concerned to decide which functions are to be implemented in hardware components and which ones in software components. This partitioning process aims at finding an optimal trade-off between conflicting requirements to improve the system performance.

Recently, different optimization methods have been undertaken to automate the hardware/software partitioning process.

These optimization methods can be split into exact and heuristic methods. The exact methods, such as Integer Linear Programing (ILP) [1], dynamic programming [2, 3], and branch-and-bound [4], work effectively for smaller graph with several tens of nodes. However, the heuristic methods produce near-optimal solutions even for larger inputs. The heuristic methods can, also, be iterative or constructive. The iterative methods such as PSO [5], Genetic Algorithm (GA) [6], Ant Colony Optimization (ACO) [7], Simulated Annealing (SA) [8], Fiduccia-Matteyeses [9], Kernighan/Lin [10], and Tabu Search (TS) [11] attempt to modify a given solution until no improvement can be done. However, the constructive methods, such as greedy and hierarchical clustering [12], generate a small number of solutions starting from an initial partitioning by selecting and adding components to the partial solution until a complete solution is obtained.

Designers of embedded systems focus on achieving more optimal partitioning solutions by emphasizing a combination between existing optimization methods. They propose to combine partitioning algorithms in order to generate optimal solutions of partitioning in a reduced time. In the literature, designers focus on combining the GA and the TS algorithms [13], the PSO and the Branch-and-Bound algorithms [14], the GA and the SA algorithms [15], and the GA and the PSO algorithms [16]. The given results prove that these combinations produce more accurate solutions than the classical algorithms in terms of cost and execution time metrics. In [17], the authors consider the reliability as a factor when solving the partitioning problem, in addition to the cost and time metrics. They propose to combine the recursive and the linear programming algorithms.

Constructive algorithms are usually suggested to be integrated with iterative algorithms to increase the quality of the generated solution. For example, the authors in [18] propose an algorithm based on clustering algorithm to make the GA algorithm better in bigger-scale embedded system. The proposed algorithm overcomes the shortcoming that GA algorithm execution time is too long to achieve good results in system partitioning.

In this work, a new hybrid method combining clustering FCM algorithm and the PSO algorithm called "FCMPSO" algorithm is proposed. Experimental results indicate that the FCMPSO algorithm is superior to GA, SA, ACO, FCM, and PSO standard algorithms and PSO-GA, GA-SA, GA-ACO, ACO-SA, FCM-GA, FCM-SA, and ACO-FCM hybrid techniques for both binary and extended partitioning approaches.

This paper is organized as follows. In Section 2, related works of the hardware/software partitioning techniques are introduced. The constructed benchmarking scenario model definition for partitioning problem is described in Section 3. In Sections 4 and 5, the formulation of hardware/software partitioning problems as a binary and then as an extended approach is presented. Experimental results and comparison of the proposed FCMPSO algorithm with standards partitioning techniques and hybrid ones are discussed in Section 6. Finally, the paper concludes in Section 7 by briefing the present work.

2. The Used Optimization Algorithms to Solve Partitioning Problems

This section provides some detailed notations and definitions of the PSO algorithm, the FCM algorithm, and our proposed FCMPSO algorithm.

2.1. PSO Algorithm. PSO is a stochastic, iterative population-based evolutionary optimization algorithm. It was developed in 1999 by Shi and Eberhart [19]. It uses the swarm intelligence that is based on social-psychological and biological social principles. By equivalence with the swarm intelligence, each swarm member (particle) takes advantage of private memory and has a degree of randomness in its movement as well as knowledge gained by the whole swarm to discover the best available food source. The problem of a food search can be solved by optimizing a fitness function. The definition of the communication structure (or social network) is obtained by assigning the neighbors for each swarm. All particles, in the search space, have fitness values which are evaluated by the fitness function to be optimized and have velocities which direct their motion in the multidimensional search space. Each particle remembers the information about its best solution and its position in the search space and both are available to its neighbors. In order to update the appropriate changes of its position and velocity, each particle p has a memory holding: the particle "p_{best}" position which presents the best solution the particle has seen by itself and the global best particle location's "g_{best}" that the particle acquires by communicating with a subset of swarms. The ith particle velocity $V_i(t)$ and position $X_i(t)$ updates are based on the following equations:

$$V_i(t+1) = wV_i(t) + C_1 r_1 \left(p_{\text{best}}(i,t) - X_i(t)\right) + C_2 r_2 \left(g_{\text{best}}(t) - X_i(t)\right), \quad (1)$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \quad (2)$$

where w is the inertia factor that takes linearly decreasing values downward from 1 to 0 according to a predefined number of iterations, $V_i(t)$ is the velocity, $X_i(t)$ is the current solution (or position), r_1 and r_2 are uniform random numbers in the range between 0 and 1, and C_1 and C_2 present positive constant parameters called "acceleration coefficient."

2.2. FCM Algorithm. FCM algorithm is a determinist, constructive optimization algorithm. Different studies prove that the FCM outperforms different existing clustering algorithms, that is, the Self-Organization Map (SOM) neural network algorithm [20], k-mean algorithm [21], and hierarchical clustering [20]. It is the most popular fuzzy clustering method, which was originally proposed by Dunn [22] and later had been modified by James [23].

FCM algorithm is efficient, straightforward, and easy to implement. It is based on fuzzy behavior and provides a natural technique for producing clustering where membership weights have a natural interpretation but not probabilistic (determinist). The main goal of the FCM is to minimize an objective function, taking into account the similarity of elements and cluster centers.

Suppose $\Omega = \{1, \ldots, k, \ldots, N\}$ a set of N objects in R^d dimensional space, listed by k. Each object k is represented by a vector of quantitative variables $X_k = \{X_{1k}, \ldots, X_{ik}, \ldots, X_{jk}\}$ defined by j variables indexed by i, where $X_{ik} \in \mathbb{R}$. Suppose $Y = \{1, \ldots, p, \ldots, C\}$ a set of C prototypes listed by p associated with C groups, where each prototype p is also represented as a vector of quantitative variables $Y_p = \{Y_{1p}, \ldots, Y_{ip}, \ldots, Y_{jp}\}$, where $Y_{ji} \in \mathbb{R}$. Suppose $U = |U_{pk}|$ a $C \times N$ matrix of membership degrees, where U_{pk} presents the membership degree of the object k to group p. Its value belongs to the real interval $[0, 1]$.

FCM algorithm aims at finding a prototype matrix Y and a membership degree matrix U that minimize $J(Y, U)$: the objective function called "fitness function." The prototypes that minimize the objective function are updated using the following equation:

$$Y_p = \frac{\sum_{k=1}^{N} \left[\left(U_{pk}\right)^m X_k\right]}{\sum_{k=1}^{N} \left(U_{pk}\right)^m}. \quad (3)$$

The membership degrees that minimize the objective function are updated according to the following equation:

$$U_{pk} = \left(\sum_{i=1}^{C} \left(\frac{\|X_k - V_p\|}{\|X_k - V_i\|} \right)^{1/(m-1)} \right)^{-1}, \quad (4)$$

where m is the level of cluster fuzziness. In the limit $m = 1$, the membership degree converges to 0 or 1, which implies a good partitioning.

FCM algorithm is an effective algorithm. It is faster than the PSO algorithm because it requires fewer function evaluations, but it is sensitive to initial values and usually falls into local optima. The weaknesses of these two algorithms motivate the proposal of an alternative approach based on the combination of FCM and PSO algorithms to form a novel FCMPSO algorithm which maintains the merits of both PSO and FCM algorithms.

2.3. FCMPSO Algorithm. In this work, FCMPSO algorithm was proposed to solve both binary and extended hardware/software partitioning problems. This algorithm takes together advantages of both PSO and FCM algorithms: the PSO algorithm has a strong global search capability, while FCM algorithm produces approximate solutions faster and fails in a local optimal solution easily. Hence, we integrated the FCM algorithm with PSO algorithm to provide near-optimal solution with faster speed.

Firstly, we applied the FCM algorithm to create the uncertain initial partitioning solutions in order to reduce (limit) the research space of the PSO algorithm. Then, we execute the PSO algorithm to have a near-optimal partitioning solution.

The pseudocode of our FCMPSO algorithm is presented in Algorithm 1.

Algorithm 1 (FCMPSO algorithm).

g_{best} = FCMPSO (Ω, C)

Require:

Dataset of FCM parameters:

(1) Select the center of cluster C

(2) Select the number of objects N

(3) Select the maximum number of iterations T

(4) Select the level of cluster fuzziness ($m = 2$)

(5) Randomly initialize U_{pk} ($p = 1, \ldots, C$ and $k = 1, \ldots, N$) of object k to group p

(6) $t \leftarrow 0$; $J(t) \leftarrow 0$; $J(t+1) \leftarrow$ Partitioning Objective Function (U_{pk})

Dataset of PSO parameters:

(1) Select the maximum number of iterations T

(2) Select the population (swarm) size: N

(3) Select the inertia factor "w" that takes linearly decreasing values downward from 1 to 0

(4) Select the uniform random numbers r_1 and r_2 in the range between 0 and 1

(5) Select the acceleration coefficients C_1 and C_2 (positive constant parameters)

FCM algorithm

Repeatedly, *while* $|J(t) - J(t+1)| > \varepsilon$ and $t < T$

(a) Update prototype matrix Y: fix the membership degree matrix U_{pk} and update prototypes using (3)

(b) Update the membership degree matrix U: fix the membership degrees using (4)

(c) $J(t) \leftarrow J(t+1)$

(d) $J(t+1) \leftarrow$ Partitioning Objective Function (U_{pk})

(e) $t \leftarrow t + 1$

end while

Return Matrices **Y** and **U**

PSO algorithm

(1) Initialize particle position X with Y matrix of FCM output algorithm

For each particle position X to D (dimension of Y)

(2) Compute the fitness values of X_i

(3) Compute velocity

(4) Initialize p_{best} to its initial position: $p_{\text{best}} = X_i$

(5) Initialize g_{best} to the minimal value of the swarm: g_{best} = fitness value (p_{best})

End For

Repeat until criteria are met ($t \geq T$)

(1) Update particle's velocity as (1)

(2) Update particle's position as (2)

If $f[X_i(t)] < f[p_{\text{best}}(I, t)]$,

(1) Update the best know position of particle i: $p_{\text{best}}(i) = X_i(t)$

If $f[X_i(t)] < f[g_{\text{best}}(t)]$,

(1) Update the swarm's best position: $g_{\text{best}}(t) = X_i(t)$

(2) $t \leftarrow t + 1$

end

return $g_{\text{best}}(t)$

The complexity analysis of the original PSO and FCM algorithm is as follows:

(i) The PSO algorithm is $O(N + 2NT)$, where N is the swarm population and T is the maximum number of iterations.

(ii) The FCM algorithm is $O(NCT)$, where N is the objects number, T is the maximum number of iterations, and C is the cluster center. In our case, $C = 1$.

(iii) The FCMPSO algorithm is $O(D + 2DT + NT)$, where D is the dimension of the population issue from the FCM algorithm, N is the objects number, and T is the maximum number of iterations.

As can be seen, the computational complexity of the FCMPSO algorithm is mainly affected by the number of objects N and the population dimension issue from the FCM algorithm. The disadvantages of the PSO algorithm are that it is easy to fall into local optima in high-population dimension and has a low convergence rate in the iterative process. It can also be observed that the computational complexity of the

hybrid FCMPSO is accepted when it is applied to solve the high-dimensional and complex problems.

Our proposed FCMPSO algorithm will be tested for two kinds of partitioning approaches: binary and extended ones. The main difference between these two kinds of architectures appears in the number of the used devices and their types. In the binary partitioning approach, the target architecture includes a single hardware processing unit and a single software processing unit or a reconfigurable architecture. However, in extended partitioning approach, the target architecture includes multiprocessing hardware components and several software processing components.

Before starting the hardware/software partitioning process, it is necessary to transform the initial specification into formal specification. The benchmarking scenario model used to validate our proposed hardware/software partitioning problem is presented in the next section.

3. Benchmarking Scenario: Task Graphs

Different benchmarks and applications are used to validate hardware/software partitioning approaches. These applications and benchmarks are varying from each other. The embedded application to be partitioned is generally given as a Direct Acyclic Graph (DAG) that represents the sequence of nodes in the embedded system application.

In this work, we use Task Graph for Free (TGFF) tool to generate a set of 20, 50, 100, 200, 500, 1000, and 2000 nodes graphs. Each graph is donated as $G(V, E)$, where $V = \{V_1, \ldots, V_n\}$ presents the set of tasks and $E\ \{E_{i,j} \mid 1 \leq i, j \leq n\}$ are the set of edges which present the data dependency between two nodes. The partitioning process aims to find a partition P, where $P = (V_H, V_S)$ such that $V_H \cup V_S = V$ and $V_H \cap V_S = \emptyset$. It can generate a deciding partition vector $X = \{X_1, X_2, \ldots, X_N\}$, representing the implementation way of the N task nodes.

Such approach is necessary to avoid the system architecture dependency variation in the system by making parameter changes. The variations in the number of the input/output nodes, the node metrics (i.e., execution time, cost, area, and power), and the software/hardware processor number are randomly assigned in the TGFF file input configuration file.

The obtained graphs are used as a system specification to validate the efficiency of our proposed partitioning algorithm to solve both binary and extended partitioning problems.

4. Hardware/Software Partitioning as a Binary Problem

This section provides the formal description of the binary partitioning problem, especially the used target architecture and the mathematical model of the objective function and the related constraints.

4.1. Architecture Representation. The characteristic of the target system architecture consists of one software processor to execute software tasks and one hardware component (FPGA/ASIC) to implement hardware tasks. Both software and hardware components have their local memory and communicate with each other through a shared bus for communication between hardware and software components, as shown in Figure 1.

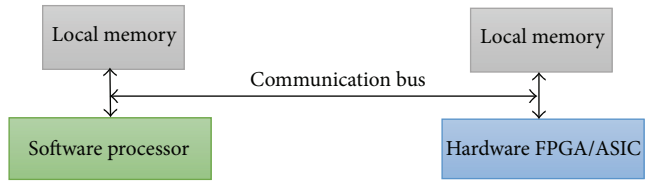

Figure 1: Target architecture of binary partitioning approach.

4.2. Model Formulation. In the binary partitioning approach, each node value must take a value of 0 or 1, where value is 1 if the node value is assigned to a hardware component; otherwise, it is 0 indicating that the task is assigned to a software component. Each node in the DAG is associated with cost parameters, that are (i) software costs (software execution time T_S, memory requirement M), (ii) hardware costs (hardware execution time T_H, used slices rate constraint S_H), and (iii) communication cost (C_{SH}). The last cost refers to the delay required to transfer data from the hardware node to software node and vice versa.

Given identical parameters and input speeds, TGFF can generate identical task graphs. These random DAG graphs allow representation of applications by using input parameters as follows.

(1) Target Architecture. One software processor and one hardware processor related over communication bus.

(2) Software Constraints. It is presented as "software execution time" constraint (T_S) fixed between 200 and 400 (μs) and "memory" constraint (M) fixed between 0 and 20 MB.

(3) Hardware Cost. It is presented as "hardware execution time" constraint (T_H) fixed between 75 and 225 (μs) and "used slice rate" constraint (S_H) fixed between 50 and 150 (slices).

The parameters generated from the TGFF input files for our three DAGs graphs are presented in Table 1.

In Table 1, AllTimeSw means the time when all nodes are implemented in software, while AllTimeHw means the time when all nodes are implemented in hardware. AllCostSw means the memory requirement when all nodes are implemented in software. AllCostHw means the hardware resource utilization when all nodes are implemented in hardware.

The communication costs required between hardware and software tasks are much less than the task processing time; it can be neglected for simplicity.

4.3. Mathematical Constraints Formulation. During the hardware/software partitioning process, based on the constraint definition in the previous subsection, the total hardware/software execution time (T_H/T_S), the used slices rate constraint (S_H), and the memory requirement (M) can be formalized as follows.

Table 1: The parameters and costs of the DAGs generated graphs.

Nodes	20 nodes	50 nodes	100 nodes	200 nodes	500 nodes	1000 nodes	2000 nodes
AllTimeSw	5761	18526	28476	58162	4230	283800	564796
AllCostSw	190	403	949	1745	142338	8463	16719
AllTimeHw	2821	7151	13842	28626	45920	139221	272813
AllCostHw	1862	4532	9727	18737	69260	92294	183506

(i) Execution Time Constraint. T_S and T_H present the execution time of, respectively, software and hardware implemented solution. They can be expressed as follows:

$$T_S(X) = \sum_{i=1}^{N} T_i^S * (1 - X_i),$$
$$T_H(X) = \sum_{i=1}^{N} T_i^H * X_i, \tag{5}$$

where N is the total number of tasks in the system and T_i^S and T_i^H are the software and hardware execution time, respectively, of each ith task. X_i presents a binary variable. Its value is 0 if the i task is assigned to a software component or 1 if the ith task is assigned to a hardware component.

(ii) The Memory Requirement. M presents the memory requirement only for components assigned to software architecture. The total memory is obtained as follows:

$$M(X) = \sum_{i=1}^{N} C_i^{Sw} * (1 - X_i), \tag{6}$$

where C_i^{Sw} is the software cost for a ith task.

(iii) The Used Slices Rate. S_H presents the number of slices that a partitioning solution will use in a particular hardware component. The total used slices rate is obtained as follows:

$$S_H(X) = \sum_{i=1}^{N} C_i^{Hw} * X_i, \tag{7}$$

where C_i^{Hw} is the hardware cost of the ith task.

Partitioning algorithms try to find a trade-off between these conflicting constraints to improve the system performance. It can be modeled as the minimization of the objective function (F.F.) as follows:

$$\text{F.F.}(\%) = 100 * \left(\frac{T_S}{\text{AllTimeSw}} + \frac{T_H}{\text{AllTimeHw}} + \frac{M}{\text{AllCostSw}} + \frac{SR}{\text{AllCostHw}} \right). \tag{8}$$

The target architecture was generally assumed to consist of only one software and only one hardware unit. In recent years, many researchers are committed to resolve the extended hardware/software partitioning problems: for multiprocessor systems with a high quality solution. In the next section, we will introduce the formal description of the extended hardware/software partitioning problem proposed in this work.

5. Hardware/Software Partitioning as an Extended Problem

This section provides the formal description of the extended architecture.

5.1. Architecture Representation. The architectural model of heterogeneous multiprocessors considered, in this section, consists of two general purpose processors and two application-specific components, denoted by W = {Sw1, Sw2, Hw1, Hw2}. Each processor has its local memory (LM) used for the intertasks communication on the same processor. Data in a shared memory is acceded by both hardware and software parts, as presented on Figure 2.

5.2. Model Formulation. In the extended partitioning problem, a node must take values between 0 and N, where N presents the number of processing units. Software nodes are performed by Sw1 and Sw2. The data communication time between tasks is much less than task processing time, and it can be omitted for simplicity. However, hardware nodes are implemented on FPGA or ASIC. Each node is associated with cost parameters that are as follows:

(i) Hardware/software execution time ($T_{Hw1}/T_{Hw2}/T_{Sw1}/T_{Sw2}$).

(ii) Used slices rate constraint (S_{H1}/S_{H2}).

(iii) Memory requirement (M_{S1}/M_{S2}) and communication time (C_{SH}).

A 20 nodes' random task graph is generated based on the TGFF using parameters denoted in Table 2.

In this work, the communication time (C_{SH}) taken between two different components was neglected.

5.3. Mathematical Constraints Formulation. In this work, the fitness function (F.F.) to minimize is defined as follows:

$$\text{F.F.}(\%) = 100 * \left(\frac{T_{Sw1}}{\text{AllTimeSw1}} + \frac{T_{Sw2}}{\text{AllTimeSw2}} + \frac{T_{Hw1}}{\text{AllTimeHw1}} + \frac{T_{Hw2}}{\text{AllTimeHw2}} + \frac{SR_{H1}}{\text{AllCostHw1}} + \frac{SR_{H2}}{\text{AllCostHw2}} + \frac{M_{S1}}{\text{AllCostSw1}} + \frac{M_{S2}}{\text{AllCostSw2}} \right), \tag{9}$$

where we define the following.

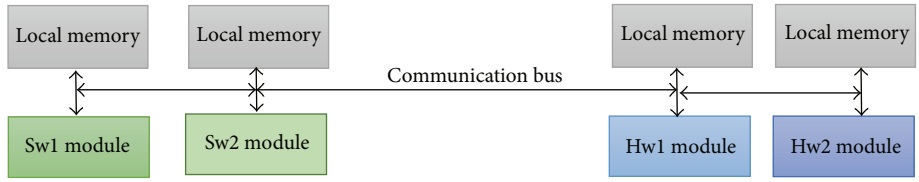

Figure 2: Target architecture of extended partitioning approach.

Table 2: TGFF parameters of partition 20 nodes with multicores architecture.

W	Node assignation	TGFF parameters		
		Execution time (sec.)	Used slices rate	Memory requirement (MB)
Sw1	"00"	T_{Sw1} [200, 400]	—	M_{S1} [0, 20]
Sw2	"01"	T_{Sw2} [100, 300]	—	M_{S2} [30, 50]
Hw1	"10"	T_{Hw1} [75, 225]	S_{H1} [50, 150]	—
Hw2	"11"	T_{Hw2} [50, 150]	S_{H2} [60, 100]	—

(i) Execution Time. T presents the time taken at each node. It can be expressed as follows:

$$\begin{aligned} T_{Sw1} &= \sum_{i=1}^{N} T_i^{Sw1} \quad \text{if } X_i = \text{"00"}, \\ T_{Sw2} &= \sum_{i=1}^{N} T_i^{Sw2} \quad \text{if } X_i = \text{"01"}, \\ T_{Hw1} &= \sum_{i=1}^{N} T_i^{Hw1} \quad \text{if } X_i = \text{"10"}, \\ T_{Hw2} &= \sum_{i=1}^{N} T_i^{Hw2} \quad \text{if } X_i = \text{"11"}, \end{aligned} \tag{10}$$

where N is number of processors. X_i presents a binary variable whose value is "00" if the i task is assigned to a Sw1 component, "01" if the i task is assigned to a Sw2 component, "10" if the i task is assigned to a Hw1 component, and "11" if the i task is assigned to a Hw2 component. T_i is the execution time for the ith task.

(ii) Used Slices Rate. S_H presents the number of slices that a partitioning solution will use in a particular hardware component. The total used slices rate is obtained as follows:

$$\begin{aligned} S_{H1} &= \sum_{i=1}^{N} C_i^{Hw1} \quad \text{if } X_i = \text{"10"}, \\ S_{H2} &= \sum_{i=1}^{N} C_i^{Hw2} \quad \text{if } X_i = \text{"11"}, \end{aligned} \tag{11}$$

where N is the total number of tasks in the system and C_i^{Hw} is the hardware cost for the ith task.

(iii) The Memory Requirement. M_S presents the memory requirement only for components assigned to software architecture. The total memory is obtained as follows:

$$\begin{aligned} M_{S1} &= \sum_{i=1}^{N} C_i^{Sw1} \quad \text{if } X_i = \text{"00"}, \\ M_{S2} &= \sum_{i=1}^{N} C_i^{Sw2} \quad \text{if } X_i = \text{"01"}, \end{aligned} \tag{12}$$

where C_i^{Sw} is the software cost for a ith task.

6. Empirical Results and Discussions

In this paper, we will evaluate the performance of the proposed FCMPSO algorithm in both binary and extended architectures. The experiments are performed on an Intel Celeron CPU having 2.16 GHz processor speed and 2 GHz RAM. Hardware/software optimization algorithms were coded in Matlab environment and they are executed in Windows 7 operating system.

6.1. Empirical Results and Discussion for Binary Partitioning Approach. The binary partitioning problem was solved considering different task graphs ranging from 20 to 2000 nodes. The values of software execution time (T_S), hardware execution time (T_H), memory requirement (M), and used slices rate (S_H) were generated randomly as described in Section 4.2.

First, we have proceed to simulate standards PSO, ACO, GA, and FCM and SA algorithms were performed. The aim is to determine the best parameters of each algorithm that can give the better solutions and compare the partitioning results. We consider the following heuristics algorithms parameters:

(i) For SA algorithm, the parameters used were as follows: initial temperature = 10, final temperature = 0, and $\alpha \geq 0.93$.

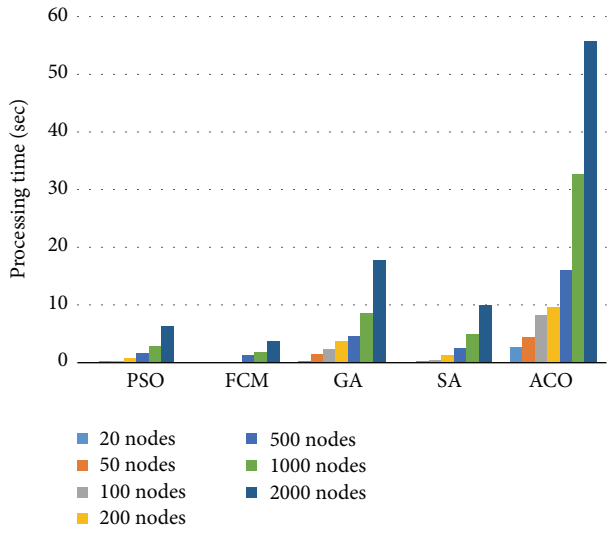

Figure 3: Processing time using standards partitioning algorithms (sec).

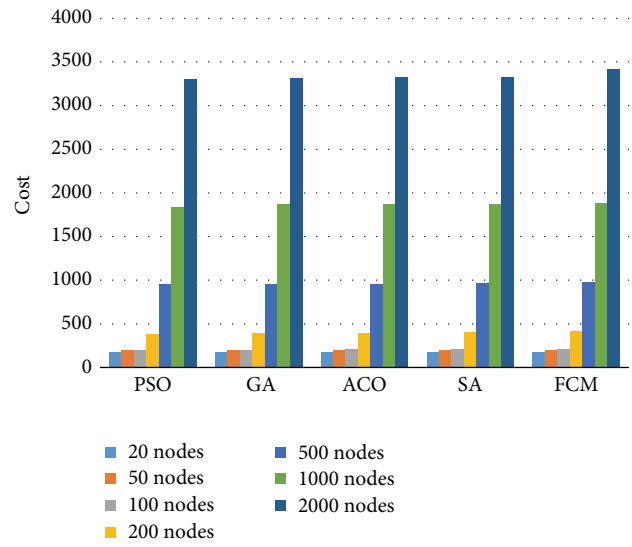

Figure 4: Costs percentage using standards partitioning algorithms.

(ii) For GA algorithm, the parameters used were as follows: population of 100 individuals, selection rate = 0.5, mutation rate = 0.2, and crossover rate = 0.5.

(iii) For PSO algorithm, the parameters used were as follows: population of 100 individuals, $C_1 \geq C_2 \geq 1$, and inertia weight $w = 0.1$.

(iv) For ACO algorithm, the parameters used were as follows: the evaporation parameter is 0.95 and the positive and negative pheromone were 0.06 and 0.3, respectively.

(v) For FCM algorithm, the parameters used were as follows: $U = [0, 2^N]$, cluster center $C = 1$, and the scalar termed $m = 2$, where N presents the number of nodes in the task graph.

Each algorithm was executed 10 times. In each execution time, the evaluations of the objective function were made 100 times. The best solutions were always taken. Performance analysis for both processing time and quality of the cost solution found by the considered algorithms is given in Figures 3 and 4, respectively.

The processing time comparison of standards partitioning algorithms, presented in Figure 3, proves that FCM algorithm is faster than PSO, SA, GA, and ACO algorithms because it requires fewer function evaluations. Figure 3 demonstrates also that the PSO algorithm takes a considerable time comparing to FCM algorithm to discover and evaluate the input random particles because of the large input swarm (50 particles).

Moreover, Figure 4 reveals that the quality result, which gives a measure of the fitness function cost percentage of solution, is found to be the best for PSO algorithm.

Results demonstrate also that PSO algorithm has a strong global search capability, while the FCM algorithm generates approximate solutions faster. Hence, a combination between the FCM and PSO algorithms allows the generation of a near-optimal solution with faster speed.

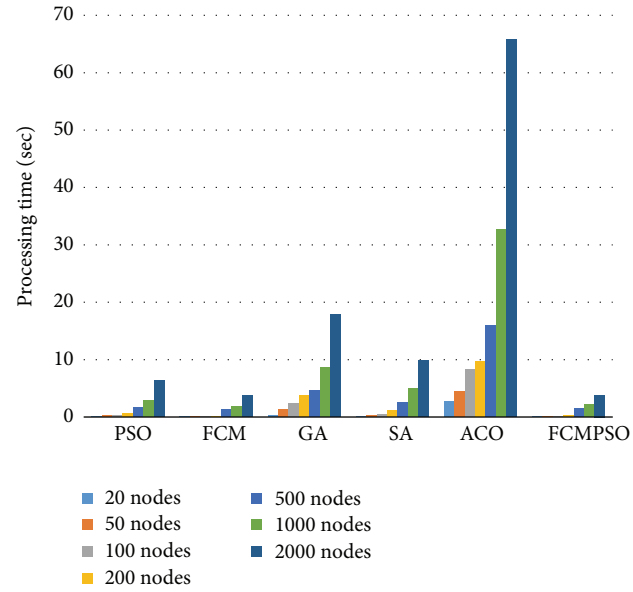

Figure 5: Comparison of processing time of FCMPSO algorithm with standards algorithms.

The simulation results of the proposed FCMPSO algorithm as well as the original FCM, PSO, ACO, GA, and ACO algorithms are presented in Figures 5 and 6.

Figure 5 reveals that the FCMPSO algorithm enhances the processing time convergence of the PSO algorithm when combining it with FCM clustering algorithm by minimizing its input population number and limiting its "local search" space.

The cost comparison provided in Figure 6 reveals the efficiency of the proposed algorithm to generate the optimal solutions comparing with the original PSO and FCM algorithms. We can also observe that the cost of generated

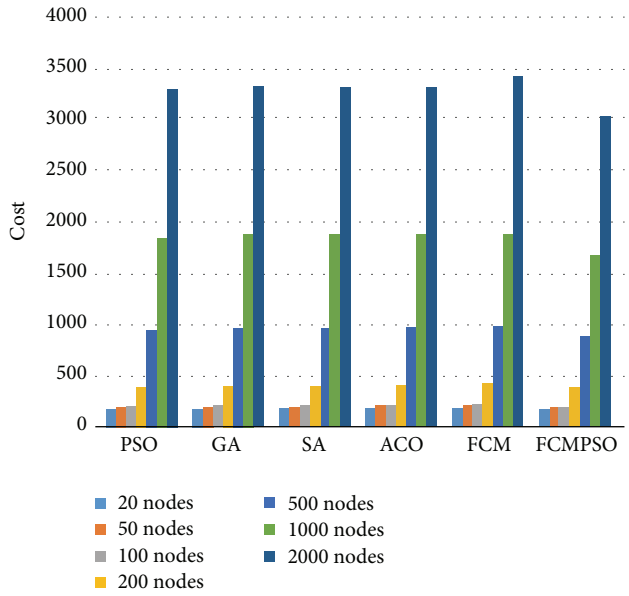

Figure 6: Comparison of cost percentage of FCMPSO algorithm with standards algorithms.

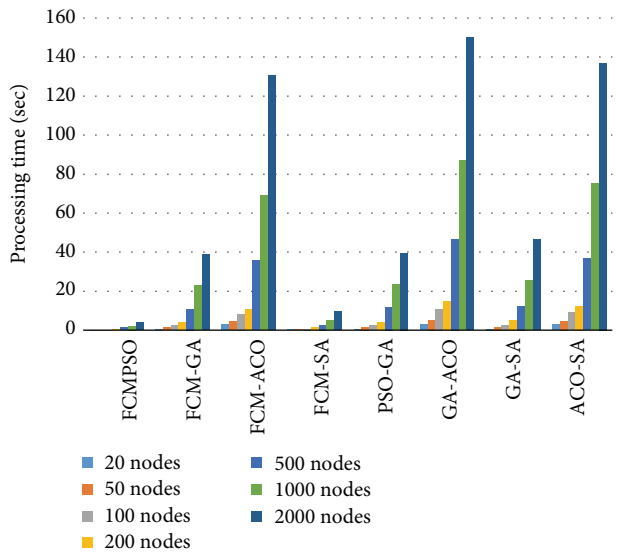

Figure 7: Comparison of processing time of FCMPSO algorithm with hybrid techniques.

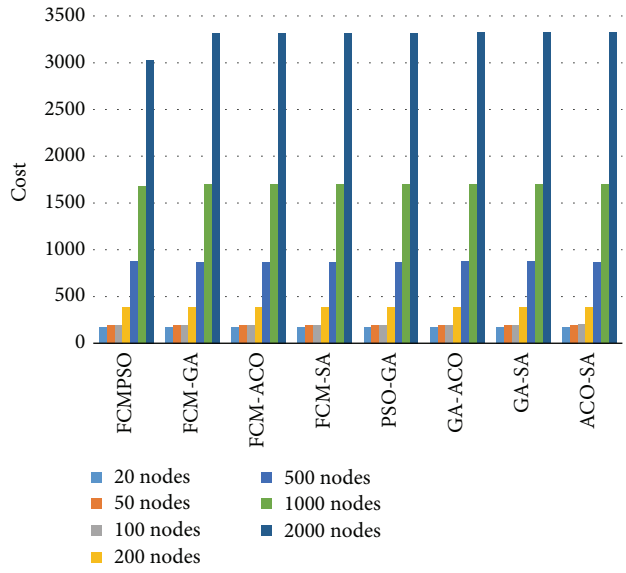

Figure 8: Comparison of cost percentage of FCMPSO algorithm with hybrid techniques.

solutions becomes more and more close when the number of nodes decreases within 2000 nodes' graph size; the variation between the FCMPSO cost and the FCM cost presents 11.11% while it is less than 4% with 20 nodes' graph size. This variation is due to the role that FCM algorithm plays in improving the convergence of the PSO by minimizing its "local search input population" to generate an improved optimal solution.

However, FCMPSO algorithm obtains significant improvements over FCM and PSO algorithms in terms of processing time and generated cost solution in binary partitioning approach.

To measure the efficiency of combining FCM and PSO algorithms, we proposed to compare it to hybrid combinations including PSO then GA, GA then SA, GA then ACO, ACO then SA, FCM then GA, FCM then SA, and finally ACO followed by FCM. Simulated results are shown in Figures 7 and 8.

the algorithm processing time of FCMPSO is 0.033 seconds while that of FCM-SA is 0.24 seconds. This result improves that around 86% of speed reduction is in favor of FCMPSO.

As shown in Figures 9, 10, 11, and 12, the best cost of FCMPSO is 170.19 while it is 170.61 for FCM-GA for 20 nodes' graphs. This result represents around 0.25% improvement in the result quality in favor of FCM-GA.

Recently, many approaches are committed to researching how to improve the performance of hardware/software partitioning in multiprocessor systems. Different solutions have been developed. In the next section, we will prove the efficiency of our proposed algorithm in an extended partitioning approach.

6.2. Empirical Results and Discussion for Extended Partitioning Approach. Extended partitioning approach tests were executed on a task graph containing 20 nodes whose parameters are described in Section 5.2. These nodes are extended for partitioning problem on multiprocessors target architecture, composed of two software units and two hardware units. Results are obtained from 10 times of execution. In each execution, the objective functions were evaluated 100 times. At each time, the best solution was used.

The simulation of the PSO, GA, ACO, SA, FCM, and our proposed FCMPSO algorithms allows the generation of individual solution, as illustrated in Table 3.

Simulation performances of both processing time and cost of the generated solution for the PSO, GA, ACO, SA,

Table 3: Generated solutions for extended partitioning approach.

	Generated solutions
FCM	01/11/10/01/01/11/01/10/10/00/11/01/00/01/01/00/00/00/00/00
PSO	01/01/10/01/11/00/00/01/10/01/01/11/00/10/11/11/00/00/00/00
GA	10/01/10/10/11/01/10/01/01/00/00/10/11/01/01/11/10/00/00/00
SA	01/01/11/00/10/01/01/10/01/10/00/10/00/00/11/11/00/00/00/00
ACO	11/10/01/10/00/00/00/01/00/00/10/11/01/01/11/01/00/00/00/00
FCMPSO	10/00/00/00/11/00/00/00/11/00/01/01/11/00/00/11/10/00/00/00

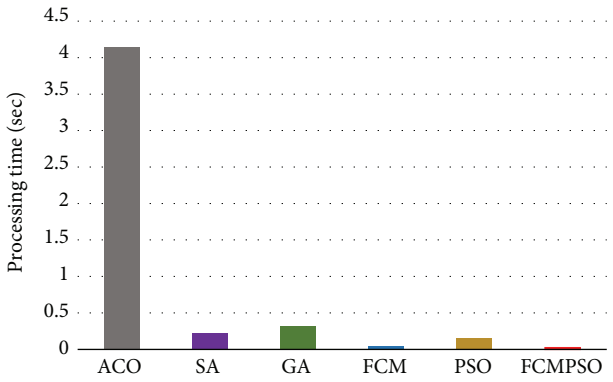

Figure 9: Comparison of processing time of FCMPSO algorithm with standards algorithms.

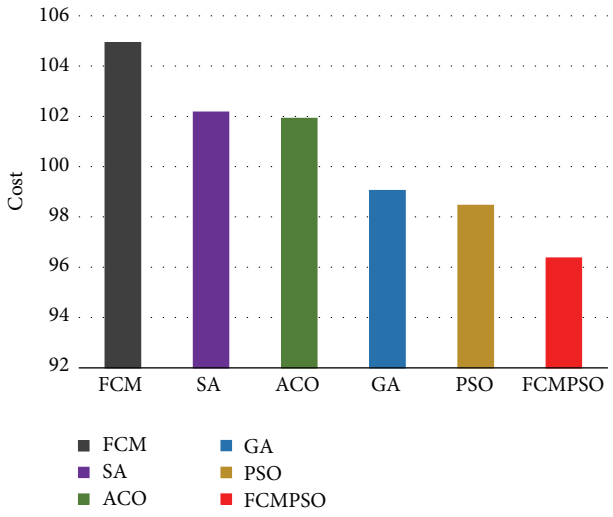

Figure 10: Comparison of cost percentage of FCMPSO algorithm with standards algorithms.

FCM, and our proposed FCMPSO algorithms are shown in Figures 9 and 10.

The obtained experimental results prove, once again, the efficiency of our proposed partitioning algorithm for an extended partitioning approach. Partitioning an embedded application on multiprocessor architecture minimizes the global cost. In fact, in our study, using two hardware units and two software units allows the decrease of global cost by 39.6% for FCM results cost (the worst cost case) and 43.28% for FCMPSO results cost (the best cost case) but took more cumulative run.

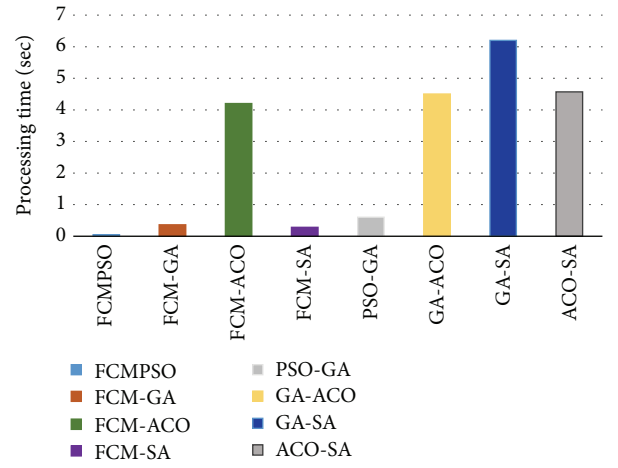

Figure 11: Comparison of processing time of FCMPSO algorithm with hybrid techniques.

To demonstrate the efficiency of our proposed algorithm, we also propose to compare it to some hybrid combinations including PSO then GA, GA then SA, GA then ACO, ACO then SA, FCM then GA, FCM then SA, and finally ACO followed by FCM. Simulated results are shown in Figures 11 and 12.

As expressed in Figures 11 and 12, the best cost was performed by FCMPSO. This result represents around 8% improvement in the result quality in favor of ACO-GA. For the algorithm processing time, the FCMPSO algorithm needs 0.051 sec while ACO-GA algorithm needs 4.57 sec. This result presents around 38% improvement in performance in favor of FCMPSO. Simulation results of both binary and extended partitioning problems demonstrate the efficiency of the FCMPSO algorithms in terms of processing time and generated solution quality.

7. Conclusion

In this paper, the FCM algorithm was integrated with PSO algorithm to form a clustering PSO called "FCMPSO." This solution maintains the merits of both PSO and FCM algorithms, to solve both binary and extended partitioning problems. FCMPSO algorithm applies FCM to the particles in the swarm to improve the generated solution (improve the quality of input swarm) in a fast processing time. We have

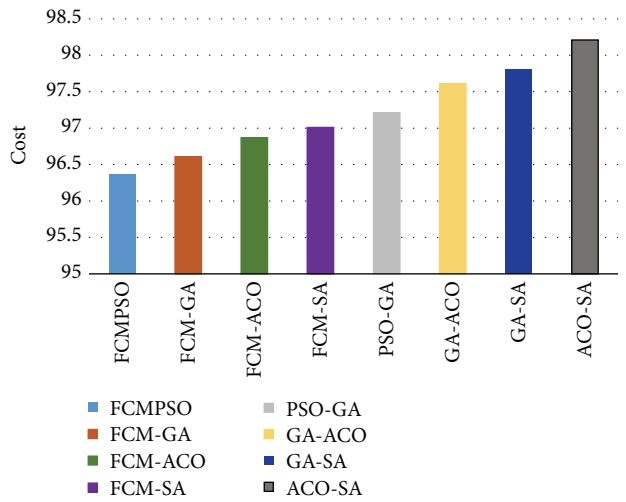

Figure 12: Comparison of cost percentage of FCMPSO algorithm with hybrid techniques.

demonstrated that FCMPSO algorithm can produce better solution with quick search speed than both standards heuristics algorithms and other hybrid partitioning techniques to solve both binary and extended partitioning problems. Our future works will consider the scheduling problem and the communication between different components.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

References

[1] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 136–156, 2005.

[2] M. O'Nils, A. Jantsch, A. Hemani, and H. Tenhunen, "Interactive hardware-software partitioning and memory allocation based on data transfer profiling," in *Proceedings of the International Conference on Recent Advances in Mechatronics (ICRAM '95)*, Istanbul, Turkey, August 1995.

[3] J. Wu and T. Srikanthan, "Low-complex dynamic programming algorithm for hardware/software partitioning," *Information Processing Letters*, vol. 98, no. 2, pp. 41–46, 2006.

[4] K. S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 193–208, 2002.

[5] A. Bhattacharya, A. Konar, S. Das, C. Grosan, and A. Abraham, "Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm," in *Proceedings of the 2nd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '08)*, pp. 171–176, Barcelona, Spain, March 2008.

[6] P. K. Nath and D. Datta, "Multi-objective hardware-software partitioning of embedded systems: a case study of JPEG encoder," *Applied Soft Computing Journal*, vol. 15, pp. 30–41, 2014.

[7] Z.-H. Xiong, S.-K. Li, and J.-H. Chen, "Hardware/software partitioning based on dynamic combination of genetic algorithm and ant algorithm," *Journal of Software*, vol. 16, no. 4, pp. 503–512, 2005.

[8] Y. Jing, J. Kuang, J. Du, and B. Hu, "Application of improved simulated annealing optimization algorithms in hardware/software partitioning of the reconfigurable system-on-chip," *Communications in Computer and Information Science*, vol. 405, pp. 532–540, 2014.

[9] J. I. Hidalgo and J. Lanchares, "Functional partitioning for hardware-software codesign using genetic algorithms," in *Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology (EUROMICRO '97)*, pp. 631–638, Budapest, Hungary, September 1997.

[10] P. K. Sahu, K. Manna, N. Shah, and S. Chattopadhyay, "Extending Kernighan-Lin partitioning heuristic for application mapping onto Network-on-Chip," *Journal of Systems Architecture*, vol. 60, no. 7, pp. 562–578, 2014.

[11] J. Wu, P. Wang, S.-K. Lam, and T. Srikanthan, "Efficient heuristic and tabu search for hardware/software partitioning," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 118–134, 2013.

[12] D. Göhringer, M. Hübner, M. Benz, and J. Becker, "A design methodology for application partitioning and architecture development of reconfigurable multiprocessor systems-on-chip," in *Proceedings of the 18th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '10)*, pp. 259–262, IEEE, Charlotte, NC, USA, May 2010.

[13] L. Li, J. Sun, W. Li, Z. Lv, and F. Guan, "Hardware/software partitioning based on hybrid genetic and tabu search in the dynamically reconfigurable system," *International Journal of Control and Automation*, vol. 8, no. 1, pp. 29–36, 2015.

[14] T. Eimuri and S. Salehi, "Using DPSO and B&B algorithms for hardware/software partitioning in co-design," in *Proceedings of the 2nd International Conference on Computer Research and Development (ICCRD '10)*, pp. 416–420, Kuala Lumpur, Malaysia, May 2010.

[15] Y. Jiang, H. Zhang, X. Jiao et al., "Uncertain model and algorithm for hardware/software partitioning," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '12)*, pp. 243–248, IEEE, Amherst, Mass, USA, August 2012.

[16] L. An, F. Jinfu, L. Xiaolong, and Y. Xiaotian, "Algorithm of hardware/software partitioning based on genetic particle swarm optimization," *Journal of Computer-Aided Design & Computer Graphics*, vol. 22, no. 6, pp. 927–942, 2010.

[17] Y. Hou, R. Wang, Y. Jiang et al., "Embedded system design with reliability-centric optimization," in *Proceedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC '15)*, pp. 33–38, IEEE, Taichung, Taiwan, July 2015.

[18] W. Li, L. Li, J. Sun, Z. Lv, and F. Guan, "Hardware/software partitioning of combination of clustering algorithm and genetic algorithm," *International Journal of Control and Automation*, vol. 7, no. 1, pp. 347–356, 2014.

[19] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the Congress on Evolutionary Computation (CEC '99)*, pp. 1945–1950, Washington, DC, USA, July 1999.

[20] S. A. Mingoti and J. O. Lima, "Comparing SOM neural network with Fuzzy c-means, K-means and traditional hierarchical clustering algorithms," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1742–1759, 2006.

[21] A. Sheshasayee and P. Sharmila, "Comparative study of fuzzy C means and K means algorithm for requirements clustering,"

Indian Journal of Science and Technology, vol. 7, no. 6, pp. 853–857, 2014.

[22] J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, no. 3, pp. 32–57, 1973.

[23] C. B. James, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Kluwer Academic, 1981.