electronics

MDPI

Article

Energy-Aware Real-Time Task Scheduling in Multiprocessor Systems Using a Hybrid Genetic Algorithm

Amjad Mahmood [1], Salman A. Khan [2,*], Fawzi Albalooshi [1] and Noor Awwad [1]

[1] Computer Science Department, University of Bahrain, Sakhir, Bahrain; amahmood@uob.edu.bh (A.M.); falblooshi@uob.edu.bh (F.A.); mis.awwad84@gmail.com (N.A.)
[2] Computer Engineering Department, University of Bahrain, Sakhir, Bahrain
* Correspondence: sakhan@uob.edu.bh; Tel.: +973-1743-7673

Academic Editor: Mostafa Bassiouni
Received: 4 April 2017; Accepted: 15 May 2017; Published: 19 May 2017

Abstract: Minimizing power consumption to prolong battery life has become an important design issue for portable battery-operated devices such as smartphones and personal digital assistants (PDAs). On a Dynamic Voltage Scaling (DVS) enabled processor, power consumption can be reduced by scaling down the operating frequency of the processor whenever the full processing speed is not required. Real-time task scheduling is a complex and challenging problem for DVS-enabled multiprocessor systems. This paper first formulates the real-time task scheduling for DVS-enabled multiprocessor systems as a combinatorial optimization problem. It then proposes a genetic algorithm that is hybridized with the stochastic evolution algorithm to allocate and schedule real-time tasks with precedence constraints. It presents specialized crossover and perturb operations as well as a topology preserving algorithm to generate the initial population. A comprehensive simulation study has been done using synthetic and real benchmark data to evaluate the performance of the proposed Hybrid Genetic Algorithm (HGA) in terms of solution quality and efficiency. The performance of the proposed HGA has been compared with the genetic algorithm, particle swarm optimization, cuckoo search, and ant colony optimization. The simulation results show that HGA outperforms the other algorithms in terms of solution quality.

Keywords: multiprocessor systems; task-allocation; task scheduling; real-time systems; genetic algorithm; power-aware task scheduling; hybridization

1. Introduction

Due to the proliferation of embedded systems (such as sensor networks) and portable computing devices such as laptops, smartphones, and PDAs, minimizing the power consumption to prolong battery-life has become a critical design issue [1–4]. Battery-operated devices rely more and more on powerful processors and are capable of running real-time applications (e.g., voice and image recognition) [4,5]. The timing requirements of real-time systems distinguish them from non-real-time systems. Contrary to non-real-time systems, a real-time system must produce logically correct results within a given deadline. Since processors consume a large amount of power in computer systems, a substantial amount of work has focused on the minimization of energy consumed by the processors. Dynamic Voltage Scaling (DVS) is one of the most effective techniques used to decrease energy consumption by a processor. DVS allows dynamically scaling both the voltage and operating frequency of processors at run time whenever the full processing speed is not required [6]. This, however, results in tasks taking longer times to complete and, consequently, some tasks might miss their deadlines. Therefore, in hard real-time multiprocessor systems, the selection of an appropriate

processor and corresponding operating voltage/frequency becomes crucial. This scenario gives rise to the power-aware task scheduling problem.

This paper focuses on the allocation and scheduling of real-time tasks in DVS-enabled multiprocessor systems. Formally, the problem addressed in this paper is defined as "*given a set of real-time tasks with precedent constraint to be executed on a dynamic voltage scaling multiprocessor system, determine the processor and the voltage level on which each task is executed such that total energy consumed is minimum, subject to tasks' deadlines and other system constraints*". The task scheduling problem is proven to be an NP-complete [7,8]. Hence finding the exact solutions for large problem sizes is computationally intractable. Consequently, a good solution can only be achieved via heuristic methods.

Even though a lot of work has been done on the scheduling of real-time tasks on distributed, uniprocessor, and multiprocessor systems, there are still extensive research efforts to develop better and efficient task allocation and scheduling algorithms under different scenarios and system requirements. In this paper, we model real-time task scheduling on DVS-enabled multiprocessor systems as an optimization problem subject to a set of constraints. The objective is to minimize the power consumption subject to task precedence and deadline constraints. To solve this problem, we propose a genetic algorithm that is hybridized with a stochastic evolution algorithm (named as HGA) to allocate and schedule real-time tasks on DVS-enabled multi-processor systems. Our approach integrates the allocation of the task to processors, scheduling of tasks on each processor, and determining the operating voltage on which a task is to be executed into a single problem. We also present specialized crossover and perturb operations as well as a topology preserving algorithm to generate the initial population. For the stochastic evolution algorithm, a number of solution perturbation schemes have been proposed to intensify the solution search. The performance of the proposed algorithm has been investigated through a comprehensive simulation/experimentation. The performance of the proposed HGA has been compared with a number of well-known metaheuristics and the results show that the proposed algorithm outperforms those metaheuristics in terms of solution quality.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 presents our system models, objective function, and constraints. The proposed algorithm is presented in Section 4 followed by simulation results and performance comparison of the proposed algorithm in Section 5. Finally, we give our conclusions in Section 6.

2. Literature Review

A number of energy efficient real-time task scheduling algorithms have been proposed in the literature both for uniprocessor and multiprocessor systems. Tavares et al. [9] used petri nets to provide a formal model for scheduling real-time tasks with strict deadlines with voltage scaling. They used pre-run time methods instead of run time scheduling methods to guarantee that all the tasks met their deadlines. Hua et al. [10] proposed a simple best-effort and enhanced on-line best-effort energy minimization strategies to schedule a set of real-time tasks. Their approaches, however, do not guarantee the completion of all the tasks within their deadlines. An approach for dynamically changing workload that integrates the DVS scheduler and a feedback controller within the earliest dead-line first scheduling algorithm is proposed in [11]. Hard real-time systems with uncertain task execution time are studied in [12]. The authors combine intra- and inter-task voltage scheduling to reduce power consumption. In the PreDVS technique proposed by Wang and Mishra [13], the processor voltage level is adjusted multiple times to attain more energy savings.

Zhu et al. [14] proposed two scheduling algorithms for tasks with and without precedence constraints running on multi-processor systems. The proposed algorithms reduce the energy consumption by reducing execution speed through the reclamation of unutilized time by a task. The scheduling algorithm proposed by Kang and Ranka [15] considered reallocating the slack for future tasks. Their algorithms work better when the execution time for all of the tasks is over- or under-estimated. Furthermore, the results show that the proposed approach is equally effective for hybrid environments in which some tasks complete before and/or after the estimated time. Kim et al. [16] use short-term

work load analysis to estimate the slack time which is then used by a preemptive rate-monotonic scheduling algorithm. A feedback scheduler for energy harvesting systems is proposed for scheduling soft real-time tasks on a DVS-enabled processor in [17]. The proposed scheduler reduces the processor's speed depending on the processor utilization and available energy in the batteries. Zhou and Wei [18] proposed a task scheduling algorithm that simultaneously optimizes the energy efficiency, thermal control, and fault tolerance for real-time systems. All of the works cited above schedule tasks on DVS-enabled uni-processor systems and hence are directly applicable to multi-processor systems where allocation of the task to the processor is also an important consideration.

For multiprocessor systems, Iterative Dynamic Voltage Scheduling (IDVS) and a Dynamic Voltage Loop Scheduling (DVLS) algorithms are proposed by Rehaiem et al. [19]. Task scheduling with deadline-miss tolerance on a system with two processors is considered in [20]. Liu and Guo [21] proposed an energy-efficient scheduling of periodic real-time tasks on multi-core processors with a Voltage Island. The authors proposed a Voltage Island Largest Capacity First (VILCF) algorithm for scheduling periodic real-time tasks. The algorithm utilizes the remaining capacity of an island prior to increasing the voltage level of the current active islands or activating more islands.

Zhang et al. [22] studied energy minimization on multiprocessor systems. Their algorithm first maps tasks to the processors and then performs voltage selection for the processors. Nelis and Goossens [23] proposed a slack reclamation algorithm to reduce energy consumption for scheduling a set of sporadic tasks on a fixed number of DVFS-identical processors. The proposed algorithm achieves energy saving by executing the waiting jobs at lower speed when a task completes earlier than its deadline. The problem of scheduling periodic and independent real-time tasks on heterogeneous processors with discrete voltage levels is discussed in [24]. They proposed an efficient heuristic for scheduling and allocating the tasks. Shin and Kim [3] proposed a two stage algorithm to determine the execution order and execution speed of tasks in a multiprocessor system. Experimental results show that their proposed technique reduces the power consumption by 50% on average over non-DVS task scheduling algorithms.

Liu [25] proposed multiprocessor real-time scheduling algorithms and efficient schedulability tests for tasks that may contain self-suspensions and graph-based precedence constraints. A genetic algorithm proposed by Lin and Ng [26] performs better than the earliest dead-line first, longest-time first, and simulated-annealing algorithms. The bat intelligence metaheuristic was used to solve the problem of reducing energy-aware multiprocessor scheduling subject to multiple objectives which are makespan and tardiness [27]. Their simulation results show a considerable improvement over the genetic algorithm in terms of solution quality. There are a number of other studies that focus on scheduling real-time task on multiprocessor systems to reduce energy consumption by dynamically adjusting the processor operating frequency [28–34].

Our proposed algorithm is different from previously proposed algorithms. We hybridized the genetic algorithm with the stochastic evolution algorithm to allocate and schedule real-time tasks with precedence constraints which has not been done before. We also proposed specialized crossover and perturb operations that exploit problem-specific solution coding, as well as a topology preserving algorithm to generate the initial population.

3. System Model and Problem Formulation

The problem considered in this paper is the scheduling of a set of real-time tasks on DVS-enabled multiprocessor systems. In this section, we present our task model, power model, and problem formulation as an optimization problem. The notations and their semantics used throughout this paper are given in Table 1.

Table 1. Notations.

Notation	Description
M	Number of processors
N	Number of tasks
v_i	Supply voltage provided to execute task $t_{i.}$
v_t	Threshold voltage
e_i	Energy consumed for processing task t_i
c_{eff}	Effective switching capacitance
l_k	Number of discrete voltage levels on which processor p_j can operate on
f_i	Clock frequency at run time (operational frequency)
T	Task set
t_i	A task in the task set, $t_i \in T$
P	Set of Processors
p_k	A processor, $p_k \in P$
τ_i	Execution time of task t_i
c_i	Number of clock cycles required to process task t_i.
pow_i	Power consumption of executing task t_i
e_{ikl}	Energy consumed by task t_i on processor pke at voltage v_l
E	Total energy consumed by all the tasks
X	An N × M matrix corresponding to a task allocation
x_{ikl}	An element of X
EST_i	Earliest start time for task t_i
LST_i	Latest start time of task t_i
FT_i	Finish time of task t_i
$Pre(t_i)$	Set of predecessors of task t_i (precedence tasks)
$Succ(t_i)$	Set of immediate successors of task t_i
ST_i	Actual start time of task $_i$
d_i	Deadline of task t_i
v_k	Set of voltage levels at which a processor p_j can operate. Voltage levels at which processor k can operate
C_1 and C_2	Acceleration coefficients in PSO algorithm
W	Constriction factor in PSO algorithm
V_{max}	Maximum velocity in PSO algorithm
A	Weight parameter for pheromone trail in ACO algorithm
B	Weight parameter for heuristic value in ACO algorithm
P	Evaporation ratio in ACO algorithm
Λ	Step size in cuckoo search algorithm
p_h	Selection probability of chromosome for perturbation

3.1. Task Model

We assume that there is a set of N real-time tasks $T = \{t_1, t_2, \ldots t_N\}$ to be executed on a DVS-enabled multi-processor system. Each task t_i is defined by (c_i, d_i) where c_i represents the worst-case computational requirement (in number of cycles) and d_i is the deadline of task t_i. We also assume that the number of clock cycles (workload) for each task is deterministic and is known a priori and the tasks are non-preemptive and hence cannot be interrupted during their execution. A task may communicate with other tasks and hence may have precedence relationships. The tasks with precedence constraints can be represented by a directed acyclic graph DAG(T, E) where T is the set of tasks and E is the set of directed arcs or edges between tasks that represent dependencies. An edge $e_{ij} \in E$ between task t_i and t_j represents that task t_i should complete its execution before t_j can start. With each edge, $e_{ij} \in E$, we assign v_{ij} that represents the amount of data transmitted from t_i to t_j. A typical task graph is shown in Figure 1.

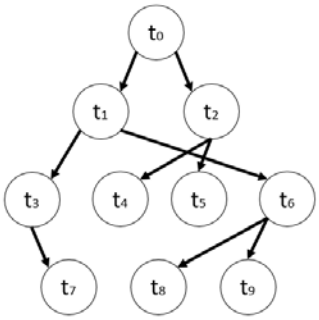


Figure 1. A directed cyclic task graph.

In our task model, we further define $Pre(t_i) = \{t_j | t_j \in T, e_{ji} \in E\}$ as a set of immediate predecessors of task t_i and $Succ(t_i) = \{t_j | t_j \in T, e_{ij} \in E\}$ as a set of tasks that are the immediate successor of t_i. If $Pre(t_i)$ is the set of predecessors of task t_i, then t_i cannot start its execution unless all of its predecessor tasks have been executed. Furthermore, $aPre(t_i) = \{t_j, t_k, t_l \ldots, t_p\}$ is a set of all the predecessors of task t_i. If $\{e_{jk}, e_{kl} \ldots e_{(p-1)p}, e_{pi} \in EandPre(t_j) = \varnothing\}$, then there is a direct path from t_p and t_i and t_p does not have any predecessor. In Figure 1, we can see that

$Succ\ (t_4) = \varnothing$ and $Succ(t_6) = \{t_8, t_9\}$.
$Pre(t_2) = \{t_0\}$ and $Pre(t_5) = \{t_2\}$.
$aPre(t_5) = \{t_0, t_2\}$ and $aPre(t_7) = \{t_0, t_1, t_3\}$.

The total execution time, τ_i, of a task t_i at frequency f_i is given by:

$$\tau_i = \frac{c_i}{f_i} \tag{1}$$

We define two auxiliary times; *Latest Start Time* (LST) and *Earliest Start Time* (EST). The latest start time of a task t_i is the time that if the task does not start by then, it will miss its deadline. If d_i is the deadline and τ_i is the execution time of task t_i, then its *Latest Start Time* is given by:

$$LST_i = d_i - \tau_i \tag{2}$$

The earliest start time of a task t_i (EST_i) is the time before which the process cannot start its execution, and is given by:

$$EST_i = \begin{cases} \max\limits_{j \in Pre(t_i)} \{FT_j\} \\ 0 \quad \text{if } Pre(t_i) = \phi \end{cases} \tag{3}$$

A task t_i will not miss the deadline if its actual execution time (ST_i) lies within its latest start time (LST_i) and earliest start time (EST_i). That is

$$EST_i \leq ST_i \leq LST_i \tag{4}$$

Consequently, the actual finish time of task t_i is given by

$$FT_i = ST_i + \tau_i \tag{5}$$

3.2. Energy Model

The processor power model used in this study has been widely used in the literature [1–3,12,22,27]. In this study, we assume a DVS-enabled multi-processor system with M processors $\{p_1, p_2, \ldots\ p_M\}$. Each processor is capable of operating on a number of discrete voltage levels. We assume that

a processor p_k has l_k discrete voltage levels. The operating voltage level of a processor can be dynamically and instantaneously changed to one of its operating voltage levels, independently of the other processors. Since processing tasks are the main contributors to energy consumption, the power consumed by a complementary metal–oxide–semiconductor (CMOS) processor is proportional to the square of the voltage applied to the circuit [6,35]. That is, if c_{eff} is the effective switching capacitance, v_i is the supply voltage, and f_i is the operational frequency (frequency at run time) on which task t_i is executed, then the power consumption is given by

$$pow_i = c_{eff} \times v_i^2 \times f_i \tag{6}$$

The relationship between power and voltage is given by

$$f_i = \xi \times \frac{(v_i - v_t)^2}{v_i}$$

where ξ is the circuit dependent constant and v_t is the threshold voltage (lowest voltage supply), and $v_i >> v_t$ generally. It is worth noting that the processor frequency decreases accordingly when the system voltage is lowered. Furthermore, the number of clock cycles required by a task t_i are known a priori and fixed, but its execution time could vary when the processor frequency changes. The energy consumed by processing a single task t_i at voltage v_i and frequency f_i (denoted by e_i) is given by [27]:

$$e_i = pow_i \times \tau_i \tag{8}$$

That is

$$e_i = c_{eff} \times v_i^2 \times c_i \tag{9}$$

According to Equation (9), the energy consumed per clock cycle is proportional to the system voltage squared. Therefore, a small change in the operating voltage of a processor can result in a significant variation in energy consumption. Thus, energy consumption can be minimized by controlling the operating voltage of the processors.

3.3. Objective Function and Constraints

Equation (9) describes the energy consumed by task t_i at voltage v_l. The total energy consumed by all the tasks, E, is then given by:

$$E = \sum_{i=1}^{N} \sum_{k=1}^{M} \sum_{l}^{|v_k|} x_{ikl}.e_{ikl} \tag{10}$$

where e_{ijk} denotes the energy consumed by task t_i when executed on processor p_j at a voltage level v_k, and x_{ijk} is a decision variable defined as:

$$x_{ikl} = \begin{cases} 1 \text{ if } t_i \text{ is allocated to } p_k \text{ at voltage level } l \\ 0 \text{ Otherwise} \end{cases} \tag{11}$$

The task scheduling problem can now be defined as a 0–1 decision problem to minimize E under certain constraints. That is, we want to

Minimize

$$\sum_{i=1}^{N} \sum_{k=1}^{M} \sum_{l}^{|v_k|} x_{ikl}.e_{ikl}$$

Subject to

$$ST_i \geq EST_i \quad \text{foreach} i,\ 1 \leq i \leq N \tag{12}$$

$$FT_i \leq d_i \quad \text{foreach} i,\ 1 \leq i \leq N \tag{13}$$

$$\sum_{i=1}^{N}\sum_{k=1}^{M}\sum_{l=1}^{|v_l|} x_{ijk} = 1 \qquad \text{foreach} i,\ 1 \leq i \leq N \tag{14}$$

The first constraint (Equation (12)) specifies that a task cannot start before the completion of all of its predecessor tasks. The second constraint (Equation (13)) specifies the real-time constraint and the last constraint (Equation (14)) specifies that each task should be assigned to exactly one processor at one voltage level.

4. Task Scheduling Using Hybrid Genetic Algorithm

The genetic algorithm (GA) [36] is one of the most powerful techniques for solving optimization problems. In order to effectively make use of the advantage of searching global spaces to find good solutions to our problem, the GA operators such as the crossover and mutation have to be altered accordingly, such that they would be applicable to the problem. In addition, the generation of the initial population consisting of feasible solutions has a large impact on the overall performance.

The hybridization of algorithms has been successfully employed to achieve better quality solutions. Hybrid algorithms are obtained by incorporating features of one heuristic into another to obtain optimal or near-optimal solutions. Hybrid approaches have generally shown better performances compared to their respective individual heuristics [37]. As far as GA is concerned, a notable amount of work has been carried out in terms of its hybridization with other heuristics such as tabu search, simulated annealing, particle swarm optimization, gravitational search, ant colony optimization, and cuckoo search [38–43]. However, hybridization of the genetic algorithm with stochastic evolution has not been reported in any previous studies, which motivates the work undertaken in our research.

Stochastic evolution [37,44,45] is a randomized iterative search algorithm inspired by the behavior of biological processes. During its execution, the algorithm maintains and operates on a single solution, and perturbs this solution in an iterative manner to incrementally increase the quality of the solution. In the classical stochastic evolution algorithm, there are two unique features; the compound move and the rewarding mechanism [37]. The compound move indicates how many individual moves (changes) are made in the current solution to reach a new solution. The compound move is analogous to multiple mutations on a chromosome in a single iteration. This is different from the classical GA where one mutation per chromosome is performed in a single iteration. The purpose of the compound moves is to allow escape from the local minima. Typically, the number of these moves should not be a high value, otherwise the new solution will be very different from its parent solution, which is not desired. Furthermore, a large compound move increases the runtime of the stochastic evolution algorithm. Thus, an appropriate value should be carefully set. In the rewarding mechanism, whenever an improvement in the solution quality is observed, the algorithm rewards itself with extra iterations, which means that additional perturbations are rewarded to the algorithm. That is, further compound moves are done on a chromosome within a single iteration The purpose of this rewarding mechanism is to allow the algorithm to search the solution space more extensively with the expectation of finding better solutions.

The pseudo-code of the proposed hybrid GA is given in Figure 2. The proposed hybrid GA starts by generating an initial population using the algorithm given in Figure 5 (line 1). The crossover operator and stochastic evolution is then applied iteratively until the termination criterion is satisfied. During each iteration, a specialized crossover operator (discussed later) is applied on selected parents (line 5). The children generated by the crossover operator are subject to a feasibility test (given in Figure 3) and infeasible children are discarded (line 7–9). A child is infeasible if at least one task in the schedule violates the deadline constraint. Stochastic evolution is then applied to a feasible child with a probability p_h (line 10–23). That is, stochastic evolution is applied to only a subset of feasible children. The stochastic evolution algorithm perturbs the solution by applying randomly selected perturb types for intensification (line 16–17). The stochastic evolution is applied for a minimum of ρ iterations which is increased if the perturb operation finds a better solution (line 21). A population replacement method

is then used to determine which solutions should be moved to the next generation (line 25). We used the roulette wheel approach for population replacement.

```
1.  Generate initial population using algorithm given in Figure 5
2.  Evaluate the fitness of the initial population
3.  while (not termination condition) {
4.     Select_parents
5.     Perform Crossover
6.     //Perform feasibility test
7.     for each child X_i
8.        if (!isFeasible(X_i))
9.         discard X_i from the population
10.    for each child X_i {
11.       if (random() <=p_h)
12.          R=Max_generation
13.          ρ=0
14.          cost=cost(X_i)
15.          do {
16.             Select a perturbation type M_t randomly
17.             X_m=Perturb(X_i, M_t)
18.             if (cost(X_m) < cost and isFeasible(X_m)) {
19.                X_i = X_m
20.                cost=cost(X_i)
21.                ρ = ρ - R }
22.             else ρ ++
23.          } while (ρ <=R)
24.    }
25.    Evaluate the fitness of children
26.    Replace the current population for the next generation
27. }
```

Figure 2. Genetic algorithm with adaptive selection of crossover and mutation.

```
booean isFeasible(chromosome X) {
1.   for (i=1;i<=N;i++)
2.     Calculate FT_i for task t_i in X
3.     if (FT_i > D_i)
4.        return false
5.  return true
6.   }
```

Figure 3. Pseudo-code to check the feasibility of a solution.

The following subsections describe the solution encoding scheme and an algorithm to generate the initial population. This is followed by a discussion on the crossover and perturbation operators used in the proposed hybrid GA.

4.1. Solution Encoding and Generation of the Initial Population

In the genetic algorithm, a chromosome represents a possible solution to the problem. Within the context of the task allocation problem, a chromosome represents the mapping of tasks to processors and voltage levels. A chromosome can be viewed as a two dimensional array with three rows and N columns, where N is the number of tasks. The first row of a chromosome represents the task number, the second row represents the processor number on which a task is to be scheduled, and the third row indicates the corresponding voltage level, as shown in Figure 4.

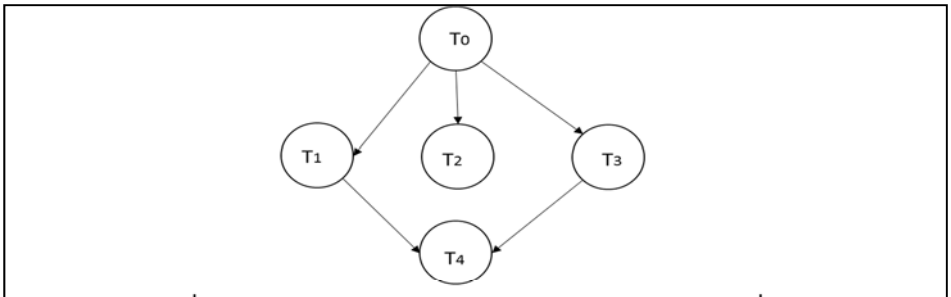
Task	1	2	3	4	5
Processor #	2	3	2	1	1
Voltage level	1	3	2	1	3

Figure 4. Solution encoding.

In order to ensure that a schedule satisfies the precedence constraint, we schedule the tasks by their topological order [46]. For this purpose, we use the algorithm given in Figure 5 to generate the initial population. The algorithm not only schedules the tasks in their topological order but also marks *segment boundaries* (tasks within a segment boundary can be executed in any order without violating the precedence constraint and therefore they can be rearranged in any sequence). These segment boundaries are required for our specialized crossover and perturb operators. The algorithm first determines all the predecessors of a task (line 6–14). It then randomly selects a task from the segment (line 20, 21) and assigns it to a randomly selected processor (line 22, 23) at a randomly selected voltage level (line 24, 25). A chromosome it added into the initial population if it satisfies the feasibility test (line 32–34) to ensure that the initial population has only the feasible solutions. Figure 6 shows two chromosomes generated by the algorithm for the given DAG (the vertical dotted lines show the segment boundaries).

```
1.  n = 0
2.  while (n<PopSize) {
3.     sNo = 0
4.     pos = 0
5.     T = TaskSet
6.     while (T ≠ ∅) {
7.        sNo++ //Segment number
8.        t_count = 0 //Number of tasks in segment
9.        sTasks = ∅        //Tasks in the current segment
10.       for each ti ∈ T
11.       if (aPre(ti) = ∅) {
12.          sTasks = sTasks ∪ {ti}
13.          t_count++
14.       }
15.       if (p=0) {
16.          boundry[l].start = pos //Segment start and end positions
17.          boundry[l].end = pos + (count -1)
18.       }
19.       while(sTasks ≠ ∅) {
20.          t = random(W)    //Select a task randomly
21.          chromosome[0][pos] = t
22.          k = random (1 .. N) //Select a processor randomly
23.          chromosome[1][pos] = k  //Assign t to processor k
24.          v = random(1 … lk)    //Select a voltage level randomly
25.          chromosome[2][pos] = v
26.          T = T – {t}
27.          sTasks = sTasks – {t}
28.          aPre(j) = aPre(j) – {t} for ∀j aPre(j) ≠ ∅ and t ∈ aPre(j)
29.          pos++
30.       }
31.    }
32.    if (isFeasible(chromosome)) {
33.       Add chromosome to population
34.       n++
35.    }
36. }
```

Figure 5. The initial solution generation algorithm.



To	T1	T2	T3	T4
4	1	1	2	3
3	1	2	1	2

To	T2	T3	T1	T4
2	1	2	0	1
1	2	3	2	2

Figure 6. Two chromosomes generated by the initial population generation algorithm.

4.2. Crossover Operator

In GA, the offspring are produced by selecting a pair of chromosomes from the current population using the roulette wheel approach and performing the crossover operation on the selected pairs of chromosomes. The crossover operator swaps corresponding randomly selected segments of two chromosomes. This ensures that the topological order in both children is preserved. Figure 7 illustrates how this crossover is performed on the second segment of the chromosomes.

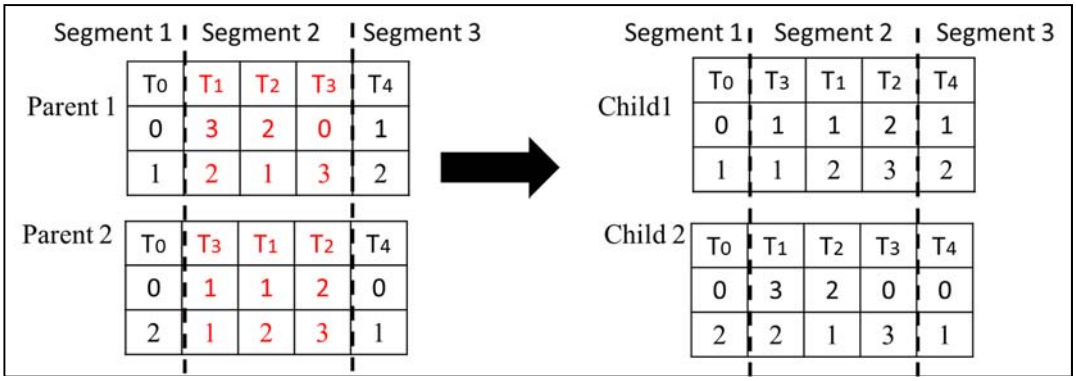


Figure 7. Crossover performed on the second segment.

4.3. Perturbation Operator

In the proposed hybrid GA, the mutation operator is replaced with the perturbation derived from the stochastic evolution algorithm. After the crossover operation is performed, a subset of chromosomes are selected for perturbation based on the probability p_h. For each selected chromosome, the perturbation operator is applied as follows. A perturbation is performed on a randomly selected gene using a compound move. In this study, we have implemented a controlled compound move for each perturbation type (described below). For types 3 and 4, a single move is performed (analogous to traditional mutation). For type 1, a compound move of size 2 is performed (changing the processor and voltage level randomly). For type 2, the size of the compound move is 4 (swapping a voltage-processor pair with another voltage-processer pair where both pairs are in the same chromosome). The cost of the resulting solution is then evaluated. If a reduction in cost is observed, then additional iterations

are rewarded to continue the perturbation. If each perturbation results in an improvement, then the algorithm keeps adding extra iterations. Perturbation is carried out until all those awarded iterations are completed. However, if no improvement is observed, then no additional iterations are rewarded. In this way, the proposed genetic algorithm implements the main functions of moves and rewarding of the stochastic evolution algorithm. In this work, four different types of perturbation are applied as explained below:

Type 1 Perturbation: In type 1 perturbation, a randomly selected gene is perturbed by changing the processor and corresponding voltage level assigned to a task. The voltage-processor pair is replaced with a randomly selected processor and voltage level.

Type 2 Perturbation: In this type, two randomly selected genes which fall in the same segment of a chromosome are swapped.

Type 3 Perturbation: In this type of perturbation, a chromosome is perturbed by selecting a gene randomly and changing the voltage level to another randomly selected voltage level.

Type 4 Perturbation: In this perturbation, a gene is randomly selected and perturbed by replacing a processor with a randomly selected processor number.

Figure 8 illustrates all the four types of perturbations. During each mutation stage of GA, we select one of the perturbation types randomly.

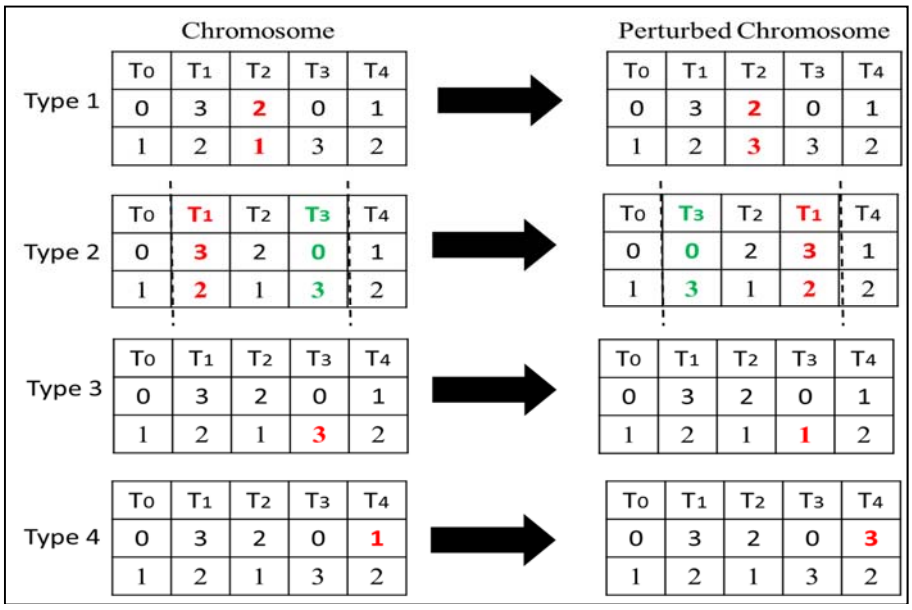


Figure 8. Different types of perturbations.

5. Results and Discussion

The performance of the HGA was evaluated empirically through comparison with other competitive metaheuristics. The algorithms were implemented in C++ on Intel i7 processor PCs with 8 GB RAM running on the Microsoft Windows 10 platform. This section discusses the experimental setup and the details of simulations done to evaluate the performance of HGA in terms of their solution quality and run time.

The basic inputs to the algorithms are DAGs of varying sizes, workload and tasks' deadlines, number of processors, and operational frequencies and voltage levels. We used both synthetic and real data of benchmarks in our experiments. For synthetic data, the TGFF utility [47] was used to generate DAGs of different sizes. The number of tasks in DAGs were between 10 and 500. The workloads for the tasks were in the range of [10, 4500] similar to the ones used in [4]. The execution time for a task on different processors was calculated by dividing the work load by the speed of the processor. We used

the technique proposed by Balbastre et al. [48] to assign the deadlines to each task in the task graph. The real benchmark data was taken from [49–51].

Results were generated for HGA as well as for other algorithms. The algorithms used for comparison were non-hybrid GA (GA), particle swarm optimization (PSO), ant colony optimization (ACO), and cuckoo search (CS). The specific variants of the metaheuristics were adapted from other relevant studies. For ACO, the variant proposed by Hyung and Sungho [33] was adapted while for PSO, the algorithm proposed by Zhang et al. [34] was adapted. Moreover, no application of CS has been reported in the literature for the underlying problem. Therefore, the basic version of the CS algorithm was implemented. For fair comparisons, the population/colony size in GA, CS, PSO, and ACO algorithms was taken to be the same as the population size of HGA that produced the best results for each test case. All algorithms were run for the same amount of time. Other parameters used for the PSO, ACO, and CS algorithms were determined after experimentation and the most suitable parameter setups are shown in Table 2. The same initial population was used for all experiments for each test case, and 30 independent runs were executed following the standard practice for statistically analyzing the performance of iterative heuristics. Results were statistically validated using the Wilcoxon-ranked-sum test at a 95% confidence level.

Table 2. Parameter settings for GA.

Algorithm	Parameter Setting
GA	pop size: 40 crossover rate: 0.8 mutation rate: 0.1
HGA	pop size: 20, 40, 60 crossover rate: 0.6 and 0.8 perturbation rate: 0.05 and 0.1 p_h: 0.1, 0.2, 0.3 rewarded iterations ρ: 3 and 5
PSO	C1 = C2 =1.49 w = 0.72 Vmax = 5
ACO	α = 2 β = 2 P = 0.2
CS	λ = 0.25

In order to find the best parameter setup for the HGA, six parameters were analyzed. These parameters are the population size, crossover rate, perturbation type, perturbation rate, chromosome selection probability p_h, and rewarded generations. The values of these parameters as used in the simulations are shown in Table 2, along with the parameters used for the other algorithms. Different arrangements of these parameter values resulted in a total of 288 combinations. Due to the computational cost involved in performing experiments for all the test cases with the 288 parameter combinations, test cases consisting of 10 and 15 DAGs were evaluated with all possible combinations to find the best parameter setup for HGA. Consequently, the following combination generally produced the best results: population size = 40, crossover rate = 0.8, perturbation rate = 0.05, p_h = 0.2, and rewarded iterations = 5. The aforementioned combination was used for subsequent experimentation with other DAGs. For GA, the parameters which produced the best results for HGA were also used for GA, as mentioned in Table 2.

Table 3 shows the cost (averaged over 30 runs) obtained by each algorithm using synthetic and benchmark real data. For synthetic data, the results indicate that for the test case with 10 tasks, all five algorithms produced results of almost the same quality. For the test case with 15 tasks, HGA and CS produced results of almost the same cost. However for all other test cases, HGA produced the best

results which are shown in Table 4 in terms of percentage improvement. The percentage improvement by HGA was in the range of 0% to over 32%. One exception was the test cases with 10 tasks where no improvement was shown by HGA. Statistical testing was also applied to all results which indicated that almost all improvements achieved by HGA were statistically significant. Based on the results, it is evident that HGA outperformed all the other algorithms considered for comparison. The conclusion is further supported by the search pattern of HGA for two test examples which are the test cases of 50 tasks and 100 tasks. Figures 9 and 10 depict a typical search pattern (for the best solution found so far during the search) for HGA and the other algorithms. It is evident from both figures that HGA was able to converge to better quality solutions faster than the other algorithms. Furthermore, the quality of the final solution obtained by HGA was better than those obtained by CS, GA, PSO, and ACO. These trends indicate the strong search capability of HGA which enabled it to produce better results. Similar trends were observed with regard to the real benchmark data. It is observed from Tables 3 and 4 that HGA was able to produce better results than the other algorithms considered, where percentage improvements obtained by HGA were in the range of 7.6% to 19.75%, and all percentage improvements were statistically significant.

Table 3. Cost comparison of HGA, GA, CS, ACO, and PSO.

Data Type	No. of Tasks	HGA	GA	Cuckoo Search	ACO	PSO
synthetic data	10	42 ± 0.00	42 ± 0.00	42 ± 0.00	42 ± 0.00	43 ± 0.80
	15	67 ± 1.49	74 ± 4.80	69 ± 3.30	75 ± 2.36	75 ± 5.68
	30	85 ± 2.68	94 ± 3.07	99 ± 4.81	89 ± 6.50	95 ± 3.37
	50	125 ± 5.09	141 ± 6.01	145 ± 5.87	152 ± 7.22	143 ± 5.99
	75	180 ± 3.77	205 ± 5.39	201 ± 7.67	223 ± 7.29	230 ± 8.15
	100	227 ± 4.96	258 ± 8.22	269 ± 8.79	271 ± 6.07	300 ± 9.49
	200	293 ± 3.99	329 ± 7.32	341 ± 6.93	352 ± 7.23	384 ± 8.02
	400	409 ± 5.03	447 ± 7.56	436 ± 6.76	430 ± 7.41	447 ± 6.29
	500	483 ± 5.70	535 ± 6.21	552 ± 5.19	516 ± 8.35	530 ± 9.01
real data [49–51]	45	92 ± 2.33	99 ± 3.76	104 ± 2.58	101 ± 2.09	107 ± 3.66
	100	170 ± 1.67	192 ± 2.32	189 ± 1.98	201 ± 3.11	199 ± 1.59
	400	319 ± 3.84	351 ± 3.09	364 ± 4.30	382 ± 2.97	346 ± 3.89

Table 4. Percentage improvement obtained by HGA with respect to GA, CS, ACO, and PSO. Statistically significant results are shown in bold.

Data Type	Tasks	HGA vs. GA	HGA vs. CS	HGA vs. ACO	HGA vs. PSO
synthetic data	10	0.00	0.00	0.00	2.38
	15	**10.45**	2.99	**11.94**	**11.94**
	30	**10.59**	**16.47**	4.71	**11.76**
	50	**12.80**	**16.00**	**21.60**	**14.40**
	75	**13.89**	**11.67**	**23.89**	**27.78**
	100	**13.66**	**18.50**	**19.38**	**32.16**
	200	**12.29**	**16.38**	**20.14**	**31.06**
	400	**9.29**	**6.60**	5.13	**9.29**
	500	10.77	14.29	**6.83**	**9.73**
real data	45	**7.61**	**13.04**	**9.78**	**16.30**
	100	**12.94**	**11.18**	**18.24**	**17.06**
	400	**10.03**	**14.11**	**19.75**	**8.46**

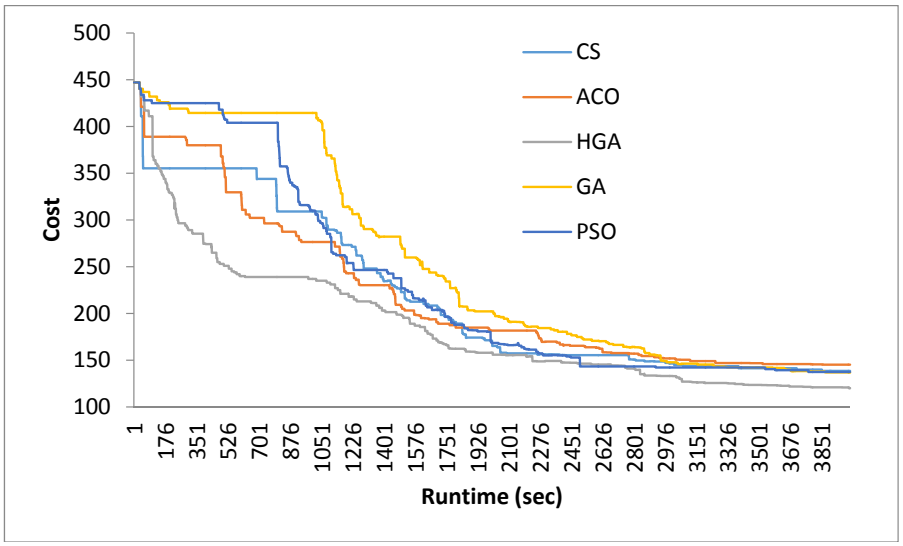


Figure 9. Plot of cost vs. runtime for 50 tasks.

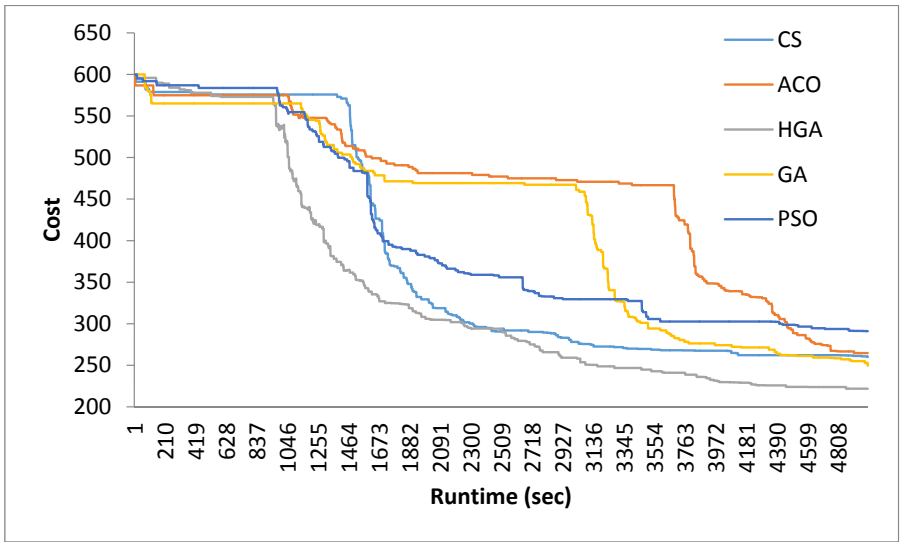


Figure 10. Plot of cost vs. runtime for 100 tasks.

To provide further insight into the search capability of the HGA algorithm, we also measured the number of invalid solutions generated by the HGA algorithm during traversal of the search space. Table 5 shows the percentage of invalid solutions for each test case. It is observed from this table that the percentage of invalid moves ranges between 3% and 14% for synthetic data, while for the real data, the range is between 8.5% and 14%.

Table 5. Percentage of invalid solutions.

	Synthetic Data									**Real Benchmark Data**		
# of tasks	10	15	30	50	75	100	200	400	500	45	100	400
% of invalid solutions	4%	3%	6%	7%	13%	11%	8%	10%	14%	8.5%	13%	14%

6. Conclusions

This paper considers the real-time task allocation problem on DVS-enabled multi-processor systems with the objective of minimizing power consumption. The problem was formulated as a combinatorial optimization problem. The paper proposed a hybrid genetic algorithm which exploits the exploration capabilities of the genetic algorithm and the intensification of the stochastic evolution algorithm. A topology preserving algorithm is used to generate the initial solution. A specialized crossover operator for GA has been proposed that does not violate the precedence constraints. Moreover, a perturb operation is defined that replaces the traditional mutation operator of the genetic algorithm. A number of specialized perturbation types have been defined for the intensified neighborhood search. A comprehensive empirical study has been carried out to evaluate the performance of the proposed hybrid GA using synthetic and real benchmark data. The performance of the hybrid GA has been compared with that of the genetic algorithm, ant colony optimization, particle swarm optimization, and cuckoo search. The results show that HGA produced better quality results as compared to the other metaheuristics. In future work, we will explore the possibility of designing the crossover and perturb operators such that invalid solutions are not generated during the search. We also intend to develop a hyperheuristic for the task scheduling.

Author Contributions: Amjad Mehmood worked on the development of the research problem, mathematical modeling of the problem, adaptation of GA for the problem studied herein. Salman A Khan worked on the development of the hybrid GA, experimental setup, analysis of the results. Fawzi Albalooshi and Noor Awwad carried out the simulation and comparisons of the algorithms. All authors contributed to writing of the paper.

Conflicts of Interest: The authors declare no conflict of interest. There were no sponsors involved in this study.

References

1. Zhuo, J.; Chakrabarti, C. Energy-efficient dynamic task scheduling algorithms for DVS systems. *ACM Trans. Embed. Comput. Syst.* **2008**, *7*, 1–22. [CrossRef]
2. Jejurikar, R.; Gupta, R. Energy-aware task scheduling with task synchronization for embedded real-time systems. *IEEE Trans. Comput. Aided Des.* **2006**, *25*, 1024–1037. [CrossRef]
3. Shin, D.; Kim, J. Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In Proceedings of the 2003 International Symposium on Low Power Electronics and Design, Seoul, Korea, 25–27 August 2003; pp. 408–413.
4. Tchamgoue, G.M.; Kim, K.H.; Jun, Y.K. Power-aware scheduling of compositional real-time frameworks. *J. Syst. Softw.* **2015**, *102*, 58–71. [CrossRef]
5. Wei, Y.H.; Yang, C.Y.; Kuo, T.W.; Hung, S.H.; Chu, Y.H. Energy-efficient real-time scheduling of multimedia tasks on multi-core processors. In Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; pp. 258–262.
6. Pering, T.; Burd, T.; Brodersen, R. Dynamic voltage scaling and the design of a low-power microprocessor system. In Proceedings of the Power Driven Microarchitecture Workshop ISCA98, Barcelona, Spain, 28 June 1998; pp. 96–101.
7. Irani, S.; Pruhs, K.R. Algorithmic problems in power management. *ACM Sigact News* **2005**, *36*, 63–76. [CrossRef]
8. Chen, J.J.; Kuo, T.W. Energy-efficient scheduling of periodic real-time tasks over homogeneous multiprocessors. In Proceedings of the PARC, September 2005; pp. 30–35. Available online: http://www.cs.utsa.edu/~dzhu/conference/parc05/papers/05-parc05-105.pdf (accessed on 4 April 2017).
9. Tavares, E.; Maciel, P.; Silva, B.; Oliveira, M.N. Hard real-time tasks' scheduling considering voltage scaling, precedence and exclusion relations. *Inf. Process. Lett.* **2008**, *108*, 50–59. [CrossRef]
10. Hua, S.; Qu, G.; Bhattacharyya, S.S. Energy reduction techniques for multimedia applications with tolerance to deadline misses. In Proceedings of the 40th Annual Design Automation Conference, Anaheim, CA, USA, 2–6 June 2003; pp. 131–136.
11. Zhu, Y.; Mueller, F. Feedback EDF scheduling exploiting dynamic voltage scaling. In Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, ON, Canada, 26–28 May 2004; pp. 84–93.

12. Xian, C.; Lu, Y.H.; Li, Z. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. *IEEE Trans. Comput. Aided Des.* **2008**, *27*, 1467–1478. [CrossRef]
13. Wang, W.; Mishra, P. PreDVS: Preemptive dynamic voltage scaling for real-time systems using approximation scheme. In Proceedings of the ACM 47th Design Automation Conference, Anaheim, CA, USA, 13–18 June 2010; pp. 705–710.
14. Zhu, D.; Melhem, R.; Childers, B.R. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Trans. Parallel Distrib.* **2003**, *14*, 686–700.
15. Kang, J.; Ranka, S. Dynamic slack allocation algorithms for energy minimization on parallel machines. *J. Parallel Distrib. Comput.* **2010**, *70*, 417–430. [CrossRef]
16. Kim, W.; Kim, J.; Min, S.L. Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis. In Proceedings of the 2003 International Symposium on Low Power Electronics and Design, Seoul, Korea, 25–27 August 2003; pp. 396–401.
17. Abbas, A.; Loudini, M.; Grolleau, E.; Mehdi, D.; Hidouci, W.K. A real-time feedback scheduler for environmental energy with discrete voltage/frequency modes. *Comput. Stand. Interface* **2016**, *44*, 264–273. [CrossRef]
18. Zhou, J.; Wei, T. Stochastic thermal-aware real-time task scheduling with considerations of soft errors. *J. Syst. Softw.* **2015**, *102*, 123–133. [CrossRef]
19. Rehaiem, G.; Gharsellaoui, H.; Ahmed, S.B. Real-Time scheduling approach of reconfigurable embedded systems based on neural networks with minimization of power consumption. *IFAC-PapersOnLine* **2016**, *49*, 1827–1831. [CrossRef]
20. Hua, S.; Qu, G.; Bhattacharyya, S.S. Energy-efficient embedded software implementation on multiprocessor system-on-chip with multiple voltages. *ACM Trans. Embed. Comput. Syst.* **2006**, *5*, 321–341. [CrossRef]
21. Liu, J.; Guo, J. Energy efficient scheduling of real-time tasks on multi-core processors with voltage islands. *Future Gener. Comput. Syst.* **2016**, *56*, 202–210. [CrossRef]
22. Zhang, Y.; Hu, X.S.; Chen, D.Z. Task scheduling and voltage selection for energy minimization. In Proceedings of the 39th Annual Design Automation Conference, New Orleans, LA, USA, 10–12 June 2002; pp. 183–188.
23. Nelis, V.; Goossens, J. Mora: An energy-aware slack reclamation scheme for scheduling sporadic real-time tasks upon multiprocessor platforms. In Proceedings of the 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Beijing, China, 24–26 August 2009; pp. 210–215.
24. Alahmad, B.N.; Gopalakrishnan, S. Energy efficient task partitioning and real-time scheduling on heterogeneous multiprocessor platforms with QoS requirements. *Sustain. Comput. Inf. Syst.* **2011**, *1*, 314–328. [CrossRef]
25. Liu, C. Efficient Design, Analysis, and Implementation of Complex Multiprocessor Real-Time Systems. Doctoral Dissertation, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2013.
26. Lin, M.; Ng, S.M. A genetic algorithm for energy aware task scheduling in heterogeneous systems. *Parallel Process. Lett.* **2005**, *15*, 439–449. [CrossRef]
27. Malakooti, B.; Sheikh, S.; Al-Najjar, C.; Kim, H. Multi-objective energy aware multiprocessor scheduling using bat intelligence. *J. Intell. Manuf.* **2013**, *24*, 805–819. [CrossRef]
28. Burmyakov, A.; Bini, E.; Tovar, E. Compositional multiprocessor scheduling: The GMPR interface. *Real Time Syst.* **2014**, *50*, 342–376. [CrossRef]
29. Chen, G.; Huang, K.; Knoll, A. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Trans. Embed. Comput. Syst.* **2014**, *13*, 111. [CrossRef]
30. Shieh, W.Y.; Pong, C.C. Energy and transition-aware runtime task scheduling for multicore processors. *J. Parallel Distrib. Comput.* **2013**, *73*, 1225–1238. [CrossRef]
31. Lipari, G.; Bini, E. A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation. In Proceedings of the IEEE 21st Real-Time Systems Symposium, San Diego, CA, USA, 30 November 2010; pp. 249–258.
32. Singh, A.K.; Das, A.; Kumar, A. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In Proceedings of the 50th Annual Design Automation Conference, Austin, TX, USA, 29 May–7 June 2013; p. 115.

33. Hyung, K.; Sungho, K. Communication-aware task scheduling and voltage selection for total energy minimization in a multiprocessor system using Ant Colony Optimization. *Inf. Sci.* **2011**, *181*, 3995–4008.
34. Zhang, W.; Xie, H.; Cao, B.; Cheng, A. Energy-Aware Real-Time Task Scheduling for Heterogeneous Multiprocessors with Particle Swarm Optimization Algorithm. *Math. Probl. Eng.* **2014**, *2014*, 287475. [CrossRef]
35. Burd, T.D.; Pering, T.A.; Stratakos, A.J.; Brodersen, R.W. A dynamic voltage scaled microprocessor system. *IEEE J. Solid-State Circuits.* **2000**, *35*, 1571–1580. [CrossRef]
36. Holland, J.H. *Adaptation in Natural and Artificial Systems*; University of Michigan Press: Ann Arbor, MI, USA, 1975.
37. Sait, S.; Youssef, H. *Iterative Computer Algorithms with Applications in Engineering*; IEEE Computer Society Press: Washington, DC, USA, 1999.
38. Zhang, A.; Sun, G.; Wang, Z.; Yao, Y. A hybrid genetic algorithm and gravitational search algorithm for global optimization. *Neural Netw. World* **2015**, *25*, 53–73. [CrossRef]
39. Kao, Y.T.; Zahara, E. A hybrid genetic algorithm and particle swarm optimization for multimodal functions. *Appl. Soft Comput.* **2008**, *8*, 849–857. [CrossRef]
40. Ciornei, I.; Kyriakides, E. Hybrid ant colony-genetic algorithm (GAAPI) for global continuous optimization. *IEEE Trans. Syst. Man Cybern. B* **2012**, *42*, 234–245. [CrossRef] [PubMed]
41. Zeb, A.; Khan, M.; Khan, N.; Tariq, A.; Ali, L.; Azam, F.; Jaffery, S.H. Hybridization of simulated annealing with genetic algorithm for cell formation problem. *Int. J. Adv. Manuf. Technol.* **2016**, *86*, 2243–2254. [CrossRef]
42. Hadded, M.; Jarray, F.; Tlig, G.; Hasni, H. Hybridisation of genetic algorithms and tabu search approach for reconstructing convex binary images from discrete orthogonal projections. *Int. J. Metaheuristics* **2014**, *3*, 291–319. [CrossRef]
43. Kanagaraj, G.; Ponnambalam, S.G.; Jawahar, N. A hybrid cuckoo search and genetic algorithm for reliability–redundancy allocation problems. *Comput. Ind. Eng.* **2013**, *66*, 1115–1124. [CrossRef]
44. Siddiqi, U.F.; Shiraishi, Y.; Dahb, M.; Sait, S.M. A memory efficient stochastic evolution based algorithm for the multi-objective shortest path problem. *Appl. Soft Comput.* **2014**, *14*, 653–662. [CrossRef]
45. Saab, Y.G.; Rao, V.B. Combinatorial Optimization by Stochastic Evolution. *IEEE Trans. Comput. Aided Des.* **1991**, *10*, 525–535. [CrossRef]
46. Oh, J.; Wu, C. Genetic-algorithm-based real-time task scheduling with multiple goals. *J. Syst. Softw.* **2014**, *71*, 245–258. [CrossRef]
47. Dick, R.P.; Rhodes, D.L.; Wolf, W. TGFF: Task graphs for free. In Proceedings of the 6th International Workshop on Hardware/Software Codesign, Seattle, WA, USA, 15–18 March 1998; pp. 97–101.
48. Balbastre, P.; Ripoll, I.; Crespo, A. Minimum deadline calculation for periodic real-time tasks in dynamic priority systems. *IEEE Trans. Comput.* **2008**, *57*, 96–109. [CrossRef]
49. Zeng, G.; Yokohama, T.; Tomiyama, H.; Takada, H. Practical Energy aware scheduling for real-time multiprocessor systems. In Proceedings of the 15th IEEE International Conference on Embedded and Real-time Computing Systems and Applications, Beijing, China, 24–26 August 2009; pp. 383–392.
50. Zhang, W.; Bai, E.; Cheng, A. Solving Energy-Aware Real-Time Tasks Scheduling Problem with Shuffled Frog Leaping Algorithm on Heterogeneous Platforms. *Sensors* **2015**, *15*, 13778–13804. [CrossRef] [PubMed]
51. Embedded System Synthesis Benchmark Suite (E3S). Available online: http://ziyang.eecs.umich.edu/~dickrp/e3s (accessed on 16 April 2017).