Multi-objective Evolutionary Algorithms for Energy-aware Scheduling on Distributed Computing Systems

Mateusz Guzek[a], Johnatan E. Pecero[b], Bernabé Dorronsoro[c], Pascal Bouvry[b]

[a] *Interdisciplinary Centre for Security Reliability and Trust, University of Luxembourg e-mail: {firstname.lastname}@uni.lu*
[b] *Faculty of Science, Technology, and Communications, University of Luxembourg e-mail: {firstname.lastname}@uni.lu*
[c] *Laboratoire d'Informatique Fondamentale de Lille, University of Lille 1, France e-mail: bernabe.dorronsoro_diaz@inria.fr*

Abstract

The ongoing increase of energy consumption by IT infrastructures forces data center managers to find innovative ways to improve energy efficiency. The latter is also a focal point for different branches of computer science due to its financial, ecological, political, and technical consequences. One of the answers is given by scheduling combined with dynamic voltage scaling technique to optimize the energy consumption. The way of reasoning is based on the link between current semiconductor technologies and energy state management of processors, where sacrificing the performance can save energy.

This paper is devoted to investigate and solve the multi-objective precedence constrained application scheduling problem on a distributed computing system, and it has two main aims: the creation of general algorithms to solve the problem and the examination of the problem by means of the thorough analysis of the results returned by the algorithms.

The first aim was achieved in two steps: adaptation of state-of-the-art multi-objective evolutionary algorithms by designing new operators and their validation in terms of performance and energy. The second aim was accomplished by performing an extensive number of algorithms executions on a large and diverse benchmark and the further analysis of performance among the proposed algorithms. Finally, the study proves the validity of the proposed method, points out the best-compared multi-objective algorithm schema, and the most important factors for the algorithms performance.

Keywords: Evolutionary Algorithms, Multi-objective Optimization, Scheduling, Energy Efficiency, Dynamic Voltage Scaling

1. Introduction

The energy efficiency of Information Technologies (IT) is one of the biggest current issues in the field of computing. The global data center power is estimated as 38.9 GW, and increased by 19% in 2012, and consecutively by 7% in 2013 [1]. The rapid increase of energy consumption of IT infrastructures, caused by growing scale and power of computing systems, resulted in development of a new discipline of IT – called commonly as GreenIT. Researchers and engineers make advances in every aspect of the domain to ensure increasing energy efficiency, with an important distinction between two categories of energy usage optimization, called static and dynamic power management [2]. The difference between them is that static power management takes place during the design time of the IT element on which it is applied. Oppositely, dynamic power management is a technique, which is executed during running of such element. One of the main dynamic management methods to optimize performance in computing systems is to use the best schedulers.

Classically, the optimal schedule is the one that executes the workload minimizing one of the execution time functions, e.g. minimizing total execution time. However, an energy-efficient scheduling algorithm has to minimize consumed energy. Such algorithms exploit power management technologies available in hardware to achieve this aim. As the biggest influence for power consumption of a server is its processor [3], this study focuses on minimizing its energy consumption. The processors manufacturers offer two main technologies: resource hibernation and Dynamic Voltage Frequency Scaling (DVFS) [2, 4].

In this work, we focus on DVFS technology that exploits the characteristics of power function of electronic circuits. The most common modern circuit technology, Complementary Metal-Oxide-Semiconductor (CMOS), has a convex power function of supplied voltage and frequency. Additionally, frequency in such circuit is linear function of voltage. Therefore, using low voltage levels leads to energy savings. This technique may also result in decreased Quality of Service (hereinafter, QoS), as decreasing processing speed may increase total execution time. However, this technique is much more adaptable to changes than resource hibernation, as DVFS transition time (30-150μs) is much shorter than hibernating a resource (few seconds, which decreases system responsiveness and thus QoS) [5]. Such elasticity of DVFS enables its direct incorporation into the scheduling process for each task. The only requirement needed to successfully and meaningfully apply that technique is that single task execution times are significantly bigger than DVFS transition time.

The most common state-of-the-art technique using DVFS is called slack reclamation [6]. It is a post-processing algorithm, which takes an already constructed schedule and uses tasks' slack times to reduce performance of processors whenever it is possible without increasing total completion time. It has been adopted in many algorithms as it is easy to apply and gives considerable energy savings (see Section 3). The main assumption that leads to the next generation of algorithms using DVFS is that involving it into the scheduling step itself will lead to greater energy savings. This work follows this direction, as used methods

allow to apply DVFS at each step of a schedule creation.

The problem of precedence constrained task scheduling is NP-hard in the simplified case of equal length tasks, homogeneous processors, and no communication costs [7]. This work tackles a generalization of the problem, with precedence constraints, heterogeneous processors using DVFS technology, communication costs, and another objective – the energy consumption.

The contributions of this paper are threefold. (1) We propose three scheduling algorithms to solve the heterogeneous multiprocessor multi-objective scheduling problem. They based on state-of-the-art multi-objective (MO) algorithms schemas, with new grouping crossover and mutation operators. (2) We do a thorough empirical study of the problem, evaluating the performance of different operators in the algorithms and the influence of instance parameters on the solutions obtained. Finally, (3) we identify the most important factors of the problem, as well as the best performing algorithm for the problem, with statistical confidence.

The remainder of the paper is structured as follows: in Section 2 the tackled problem is described. The state of the art on energy-aware scheduling is presented in Section 3. Section 4 describes the proposed solution for the problem and Section 5 analyses the results of simulations over the large set of instances. Section 6 includes conclusion and presents future research directions.

2. Problem Description

The addressed scheduling problem deals with the optimal allocation of a set of tasks that compose a parallel application to the set of processing elements in a distributed system. The target is minimizing both the total execution time and energy consumed by the execution of the application.

2.1. System

The distributed computing system consists of a set R of m heterogeneous processors. For each processor r_l, a set of DVFS pairs D^l is defined. A DVFS pair k, denoted as d_k, is a tuple (v_k, s_k), where $v_k \in \mathbb{R}^+$ is the operational voltage of the processor and $s_k \in \mathbb{R}^+$ is the ratio of the operational speed to the maximal processor's speed, further called as relative speed.

We consider a parallel application represented by a Directed Acyclic Graph (DAG) $G = (T, E, p_{il}, c_{ij})$, where T denotes the set of t tasks, and the set E of directed edges indicates the precedence relation between tasks. Each node $n_i \in T$ is associated with one non-divisible task t_i of the parallel application. The processing time or weight $p_{il} \in \mathbb{R}^+$ of task t_i denotes its computation time on processor $r_l \in R$. For each edge $e_{ij} \in E$ there is a cost $c_{ij} \in \mathbb{R}^+$ to communicate data between tasks t_i and t_j. The partial order $t_i \prec t_j$ models precedence constraints, i.e. if there is an edge $e_{ij} \in E$. In this case, t_i is said to be an immediate predecessor of t_j, and $\Gamma^+(t_j)$ denotes the set of immediate predecessors of t_j. Also, t_j is said to be an immediate successor of t_i, and $\Gamma^-(t_i)$

represents the set of immediate successors of task t_i. A task t_i without predecessors, $\Gamma^+(t_i) = \emptyset$, is called an entry task. If it does not have any successors, when $\Gamma^-(t_i) = \emptyset$, it is a sink task.

A scheduling is defined as a tuple of functions $(\sigma, \pi, \lambda, \gamma)$, such that $\sigma : T \mapsto \mathbb{R}^+$, $\pi : T \mapsto R$, $\lambda : T \mapsto D^l$, and $\gamma : R \times X \mapsto D$, where $\sigma(t_i)$ represents the time at which task t_i starts its execution, $\pi(t_i)$ provides the processor r_l on which t_i is executed, $\lambda(t_i)$ supplies the relative speed s_{li} to run t_i, and $\gamma(r_l, x)$ provides the DVFS pair d_k used by the processor l at time x. The real execution time of a task i scheduled on a processor l with a DVFS pair k is defined as: $p_{ilk} = p_{il}/s_k$.

2.2. Constraints

A schedule is feasible if the following conditions are fulfilled for all tasks in G.

1. The Processor Constraint. For any pair of tasks $t_i, t_j \in T$, one processor may not execute more than one task at a time. That is, if $\pi(t_i) = \pi(t_j) = r_l$, then $\sigma(t_i) + p_{il\lambda(i)} \leq \sigma(t_j)$ or $\sigma(t_j) + p_{jl\lambda(j)} \leq \sigma(t_i)$.
2. The Precedence Constraint. $\forall\ (e_{ij} \in E)$, the task t_j cannot be executed before task t_i has finished its execution: $\sigma(t_j) \geq \sigma(t_i) + p_{il\lambda(i)}$ if $\pi(t_i) = \pi(t_j)$. Otherwise, if $\pi(t_i) = r_k$ and $\pi(t_j) = r_l$ and $r_k \neq r_l$, $\sigma(t_j) = max_{t_i \in \Gamma^+(t_j)} \sigma(t_i) + p_{ik\lambda(i)} + c_{ij}$, the time when all communications from t_j's predecessors have arrived at r_l.
3. The DVFS Constraint. For any time, a processor r_l has allocated exactly one DVFS pair, i.e. $\forall\ (x \in X)\ \forall\ (r_l \in R)\ \exists!\ (d_k \in D^l)$ such that $\gamma(r_l, x) = d_k$, where X represents the time when the system is used.

2.3. Objectives

The completion time of each task $t_i \in T$ is defined by $C_{t_j} \equiv \sigma(t_j) + p_{jl\lambda(t_j)}$. The makespan of the schedule is defined as: $C_{max} \equiv \max_{t_j \in T} C_{t_j}$ and is the maximum completion time of a task of G. The first objective of the optimization is to find a schedule for G on the processors R with the minimum makespan: $\min C_{max}$.

The second objective is related to the energy consumed by the system. It is based on the equation of dynamic power consumed by the CMOS circuit (the prevalent technology used in modern integrated circuits) [6]:

$$P = AC_{ef}V^2 f = \alpha V^2 f, \tag{1}$$

where A is the number of switches per clock cycle, C_{ef} is the total capacitance load, V is the supplied voltage, and f is the corresponding frequency. Relative speed rs_j is proportional to frequency f, so we use directly rs_j as reported by processor manufacturers instead of f. As A and C_{ef} are constant for a machine, we simplify them to single coefficient α. The value of α is always set to 1 to normalize the voltage-frequency tables. Finally the consumed energy is defined as:

$$E_t = \sum_{j=0}^{m} \int_0^{C_{max}} P_l(x)dx = \sum_{l=0}^{m} \int_0^{C_{max}} v_l(x)s_l(x)^2 dx, \quad (2)$$

where x is time, $P_l(x)$ is the power of machine r_l over time, $v_l(x)$ is the voltage of machine r_l over time, and $s_l(x)$ is the relative speed of the machine r_l over time. The second objective is to minimize the consumed energy: $\min E_t$.

The problem of scheduling DAGs with minimum makespan and energy is a trade-off between schedule length and energy consumption. The reduction in energy consumption in DVFS-enabled processors is made by decreasing supply voltage and frequency, resulting in a slower tasks execution and an increase in the schedule length. As energy is a convex function of relative speed, running a task using lower frequency results in lower total energy consumption.

3. Energy-aware Scheduling: Related Work

An important number of scheduling algorithms have been proposed for energy consciousness. These algorithms differ on the assumptions they consider [4]. However, the most common technique to save energy they exploit is DVFS with slack sharing or slack reclamation.

Zhu et al. present in [8] slack reclamation-based scheduling algorithms. The algorithms adopt a global scheduling in which all tasks wait in a global queue and are dispatched based on their priorities. Zhang et al. [9] report a scheduling algorithm based on a two-phases process. The first phase aims to optimize the possibilities for selecting different voltages based on the priority of tasks. Then, the voltage scaling problem is modeled as an integer programming problem in the second phase. The authors considered continuous supply voltage selection and showed that the integer programming problem is solvable in polynomial time. To solve the discrete version of the problem, the authors proposed an approximation algorithm.

Aupy et al. [10] study the problem of scheduling precedence-constrained applications aiming to minimize energy consumption while considering a given bound on the makespan and a reliability threshold. The target architecture is a set of homogeneous processors. The authors propose several polynomial time scheduling algorithms under the continuos speed model.

Wang et al. exploit the idea of extending the execution time of non-critical jobs by lowering the speed of processors without extending the makespan's application [11]. Two scheduling algorithms are proposed. The first algorithm exploits the best-effort idea: it first optimizes makespan under maximum voltage and speed assumption using a list scheduling algorithm and then energy is optimized in a second step with a voltage scaling algorithm, by lowering the processor's voltage for extending the execution time of non-critical jobs, or when it is in idle time. The second algorithm is a clustering-based scheduling algorithm that gather tasks into clusters according to the edge zeroing policy. It is guided with aim of reducing power consumption.

Baskiyar and Palli [12] use the Heterogeneous Earliest Finish Time (HEFT) heuristic as a best-effort scheduling algorithm, then it performs voltage scaling without performance degradation. The authors considered continuous voltage. In [13] the authors combined the Decisive Path Scheduling (DPS) list scheduling algorithm and dynamic voltage scaling with dynamic power technique.

Rizvandi et al. report in [14] a heuristic called Multiple Frequency Selection DVFS. The algorithm exploits the idea of executing tasks using a linear combination of processors' frequencies so that the utilization of all slack times is optimized. For each task its energy consumption is formulated as a constrained optimization problem. Then, the authors show that a combination of two frequencies lead to minimum energy consumption.

Some recent approaches incorporate DVFS during the scheduling process. Shekar and Izadi develop in [15] an algorithm that schedules tasks to processors with low-power capability. The authors proposed a weighted cost function that considers the energy consumptions while taking scheduling decisions. Lee and Zomaya [6] report a set of Dynamic Voltage Scaling (DVS)-based heuristics to optimize a summation of two objectives: schedule length and energy consumption. After the heuristic computes the schedule of the application the arrangement is improved by using a local search algorithm. The local search only applies changes if it does not increase the schedule length and energy is minimized.

Some researchers address the energy issue in a Pareto based approach. Mezmaz et al. proposed in [16] a parallel bi-objective hybrid genetic algorithm that is improved with the heuristics reported in [6]. The parallelization is based on the cooperative approach of the island and multi-start parallel models using the farmer-worker paradigm. The goal of the multi-start parallel model is to reduce the running time of a resolution. Pecero et al. developed in [17] a bi-objective Greedy Randomized Adaptive Search Procedure (GRASP). The GRASP algorithm starts by generating a feasible solution using a greedy evaluation function at maximum voltage. Then, the solution is improved by a post-processing bi-objective local search. The bi-objective local search exploits the DVS technique.

4. Algorithms Description

This section presents the algorithms used to solve the considered problem in this study. They are three state-of-the-art algorithms with different features that have been specialized for the problem at hands with a novel representation and operators. The source code of the implementation is published as an open source greenMetal project and available online[1].

4.1. MOEAs schemas

The chosen three state-of-the-art techniques to find accurate solutions to the investigated problem are: NSGA-II [18], MOCell [19], and IBEA [20].

[1] http://greenmetal.gforge.uni.lu/download.html

Pseudocode of MOCell

```
1: //Algorithm parameters in 'mocell'
2: InitializeParetoFront(mocell.ParetoFr)
3: while ! StopCondition() do
4:     for ind ← 1 to mocell.popSize do
5:         n_list←GetNeighb(mocell, ind);
6:         parents←Selection(n_list);
7:         offspr←Recomb(parents);
8:         offspr←Mutation(offspr);
9:         Evaluation(offspr);
10:        Insert(offspr,aux_pop[ind]);
11:        InsertParetoFront(ind);
12:    end for
13:    mocell.pop←aux_pop;
14:    mocell.pop←Feedback(mocell,ParetoFr);
15: end while
```

Pseudocode of IBEA (Adaptive Version)

```
1: //Algorithm parameters in 'ibea'
2: InitializePopulation(ibea.pop);
3: EvaluationUsingHypervolume(ibea.pop);
4: while ! StopCondition() do
5:     union←Merge(ibea.paretoFr,offsprPop);
6:     EvaluationUsingHypervolume(union);
7:     RemoveWorstIndivs(ibea.paretoFr);
8:     while offsprPop ≤ ibea.popSize do
9:         parents←Selection(ibea.paretoFr);
10:        offspr←Crossover(parents);
11:        offspr←Mutate(offspr);
12:        EvaluationUsingHypervolume(offspr);
13:        offsprPop←Add(offspr);
14:    end while
15: end while
```

Pseudocode of NSGA-II

```
1: //Algorithm parameters in 'nsga'
2: InitializePopulation(nsga.pop);
3: Evaluation(nsga.pop);
4: while ! StopCondition() do
5:     for index ← 1 to nsga.popSize/2 do
6:         parents←Selection(nsga.pop);
7:         offspr←Crossover(parents);
8:         offspr←Mutate(offspr);
9:         offspringPop←Add(offspr);
10:    end for
11:    Evaluation(offspringPop);
12:    union←Merge(nsga.pop, offspringPop);
13:    fronts←SortFronts(union);
14:    (Pop', lastFront)←GetBestFronts(fronts);
15:    if size(nextPop) < nsga.popsize then
16:        Pop'←BestAccToCrowding(lastFront,
               nsga.popsize-size(Pop'));
17:    end if
18: end while
```

Figure 1: Pseudocode of the studied multi-objective evolutionary algorithms.

Their pseudocodes are shown in Figure 1. The selection was guided to include algorithms with diversified features, like solutions ranking, feedback of non-dominated solutions from the archive into the population, or indicator based search, in which the *quality indicator* is a function which maps a Pareto set approximation into a real number [20]. Therefore, the search process is guided by a metric representing quality of a solution instead of its value.

NSGA-II [18] is, probably, the most referenced algorithm in MO optimization. It uses a dominance depth ranking, which means that solutions are ranked according to the order of Pareto front to which they belong. The whole set of solutions is divided into Pareto fronts in recursive manner: first, the best Pareto front is created. After that, this front is removed from the population and the next Pareto front is created from the remaining solutions. Each consecutive front is one order higher than the previous one. The whole population is classified in this way and then fitness is distributed according to the order of solutions –the lower the order, the higher the fitness. Those solutions with higher fitness are preserved for the next generation. If there are additional solutions with the same fitness, crowding distance is used to preserve diversity.

MOCell [19] uses external archive and spatial division of solutions. The archive is used to store the best found solutions as well as to provide feedback,

i.e., to randomly replace solutions from the main population with solutions from the archive. The spatial division is done as in the canonical cellular GA [21]: each individual has its own place in a toroidal mesh and only the solutions from the certain neighborhood have chance to compete during the selection procedure. MOCell uses typical C9 neighborhood (including solution and its 8 closest neighbors). The spatial distribution of population provides more exploration possibilities by creating distinct areas, where different good solutions may emerge and by slowing down the convergence. When the archive is full, the crowding distance is used to remove the solutions from the most crowded regions.

IBEA [20] implements hypervolume, a quality indicator, to assign fitness to solutions. Each solution is compared with all others in the population by hypervolume and its final fitness is set according to its aggregated performance against the others. Then, during the environmental selection phase, the worst solutions are removed until population reaches its allowed size.

Despite all mentioned differences, all three algorithms share large part of their characteristics. They all use binary tournament selection as mating selection mechanism, which creates new individuals for the population. In the implementations in this work, they share the representation (presented in Section 4.2), the grouping recombination operator, and the bit-flip mutation (both described in Section 4.3). The probability for crossover and mutation has the same influence for all of them. Finally, they use the same concepts of individual, population, generation, and evaluation.

The selected algorithms ensure that different state-of-the-art approaches were used to explore the problem, to avoid possible biased or unexpected behavior of one of the algorithms. Using the same parameters and operators of the algorithms ensures a fair comparison and is intended to neglect the impact of auxiliary factors. Additionally, the algorithms schemas were implemented using the jMetal [22] framework which was extended by a new GreenMetal set of libraries, which implements the studied MOP as well as the representation and operators described in further sections.

4.2. Representation

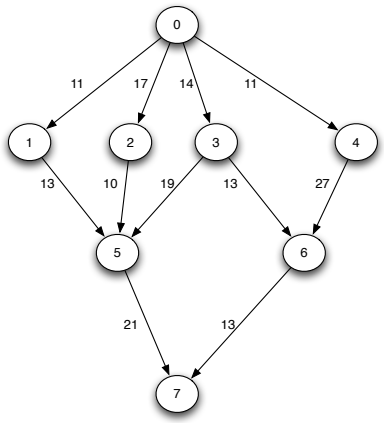
There exist three main groups of representing multiprocessor scheduling problems [23]: node list, processor allocation, and direct representation. In the first approach only the task priority is given, in the second one each task is assigned to a machine. These two representations must be combined with an external method that will create the final schedule. The direct representation includes both assignment and the priority, resulting in the exact mapping, however at expense of increasing complexity, and complex genetic operators.

The representation of a solution designed for this work is based on the processor allocation approach. A solution is composed of two vectors: the first contains information about the processors allocation, while the second one determines the DVFS pair that must be used by the allocated processor to run the task.

Table 1: Sets of sample DVFS pairs of three heterogeneous processors [6].

Level	Type 1		Type 2		Type 3	
	V_k	RS_s (%)	V_k	RS_s (%)	V_k	RS_s (%)
0	2.20	100	1.50	100	1.75	100
1	1.90	85	1.40	90	1.40	80
2	1.60	65	1.30	80	1.20	60
3	1.30	50	1.20	70	0.90	40
4	1.00	35	1.10	60		
5			1.00	50		
6			0.90	40		

The usage of the processor allocation model is motivated by the possibility of the usage of the structure of the DAG during the schedule creation, which guides the algorithm that sets the tasks priority. While restricting the set of available solutions, this approach removes regions of search space with clearly infeasible or low quality solutions. This restriction is additionally useful, as the problem becomes more complex because of adding the second vector of the decision variables to the solution representation. The vectors length is equal to task number and a greedy heuristic is used to create final scheduling. Examples in this section are based on the application presented in Figure 2, which is run on processors presented in Table 1. Note that processors may have different available DVFS pair, which may have different mapping to used voltage and relative speed.



task	r_0	r_1	r_2	$\overline{p_i}$	b-level
0	11	13	9	11	101.3
1	10	15	11	12	66.7
2	9	12	14	12	63.3
3	12	16	10	12	73.0
4	15	11	19	15	79.3
5	13	9	5	9	41.7
6	11	15	13	12	37.3
7	11	15	10	12	12.0

Figure 2: On the left, a sample DAG with the task indexes i inside nodes and values of $c_{i,j}$ function next to the corresponding edges. On the right, computation cost (p_i at level L_0) and task priorities (b-level).

A sample solution encoded in this form is presented in Figure 3. The first row shows the position of a gene, which binds this gene to a task with a given number. The second row represents the processor assignment, by indicating the processor identifier to which a task is assigned. The third row determines the

voltage level on the indicated processor. For instance, we can see in the figure that task 7 is assigned to processor 2 with the DVFS pair 3.

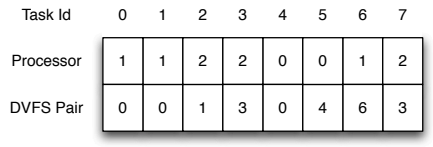


Figure 3: Example of utilization of the proposed processor assignment representation. Each gene corresponds to assignment of one task.

The addition of the DVFS pair component makes the search space growing exponentially, which in case of DVFS-disabled system is m^t (for m machines and t tasks), while in DFVS-enabled system it is $(\sum_{j=0}^{m} pairs(j))^t$ ($pairs(j)$ represents the number of available DVFS pairs for machine j) [24].

4.3. Operators

There are two operators, which were implemented in problem specific form: crossover and mutation. These operators are explained in detail in the following sections.

4.3.1. Grouping Recombination

The crossover selected for this study is a grouping crossover [25, 26]. It does not work on the tasks alone, but rather on groups of tasks. Such crossover is also able to merge whole groups of tasks, working on higher level than simple single or double point crossovers. As a result it is more likely that the whole groups of allocated tasks will merge and be executed on a single machine, which consequently leads to the decreased communication time and makespan minimization. The selected DVFS pair is not modified by this operator.

In the first stage, the operator randomly selects a subset of all processors used in a solution. It is done by using an auxiliary group part for each solution. A group part is a random permutation of the list of processors used in a solution. The pseudo code of low complexity group part generation function is presented in Algorithm 1. The loops execution numbers are bounded by processor number m or length of solution equal to task number t. Complexity of this algorithm is therefore $O(t + m)$, which can be simplified to $O(t)$ under the assumption that $t > m$.

After group part generation, a random point is selected in the group part and processors from this point until the end of group part form the subset used further in the crossover operation. All genes with assignments to the processors from this subset are then copied from the second parent to the first one. The pseudo code of this operator is presented in Algorithm 2. The length of group part and consequently the number of repetitions of outer loop is equal to m and inner loop always iterates t times. Complexity of this algorithm is $O(tm)$.

Algorithm 1 Group part generation

```
Require: solution S, processor number proc_num
aux_array ← create_array(integer,proc_num);
for all i in 0 :get_length(aux_array)−1 do
    aux_array[i] ← −1;
end for
processors← 0;
for all i in 0 :get_length(S)−1 do
    gene ← get_gene(S, i);
    pa ← get_processor_assignment(gene);
    if aux_array[pa] == −1 then
        aux_array[pa] ← pa;
        processors ← processors +1;
    end if
end for
group_part ← create_array(integer,processors);
j ← 0;
for all i in 0 :get_length(aux_array)−1 do
    if aux_array[i] ! = −1 then
        group_part[j]← aux_array[i]);
        j ← j + 1;
    end if
end for
for all i in 0 :processors−1 do
    r ←random(i, processors);
    swap(group_part[i],group_part[r]);
end for
return group_part;
```

A sample operation of the crossover is presented in Figure 4. Offspring is initialized as a copy of a first parent. The group part is generated from the second parent. A randomly chosen crossover point separates a section from the group part, which has in this case one element: 2. Then, all occurrences of separated elements in the group part are copied to the offspring on the same position as they occur in second parent (1,2,4,5).

4.3.2. Mutation

The two proposed mutation operators are in general forms of bit-flip mutation, adapted to the problem. As the grouping recombination work on high-level, the selected mutation is simple and intended to introduce random behavior, maintaining the diversity in the population. In the first type, both processor and DVFS pair assignments are changed. As processor determines valid DVFS pairs, it is selected randomly in the beginning. The processor may be changed to any other processor in the system. After that, the DVFS pair is chosen among the ones available for the processor. In the second type, only the DVFS pair assignment is changed. The DVFS pair may be changed only to one available for the given processor. New values are chosen randomly using a uniform dis-

Algorithm 2 Grouping crossover operator

Require: parent solutions: S_1, S_2
1: $S'_1 \leftarrow S_1$;
2: $group \leftarrow generate\_group\_part(S_2)$;
3: $gl \leftarrow get\_length(group)$;
4: $r \leftarrow random(0, gl-1)$;
5: **for all** i in $r : gl-1$ **do**
6: **for all** j *in* $0 : get\_length(S'_1)-1$ **do**
7: $tmp\_gene \leftarrow get\_gene(S_2, j)$;
8: **if** $get\_processor\_assignment(tmp\_gene) == group[i]$ **then**
9: $set\_gene(S'_1, j, tmp\_gene)$;
10: **end if**
11: **end for**
12: **end for**
13: **return** S'_1;

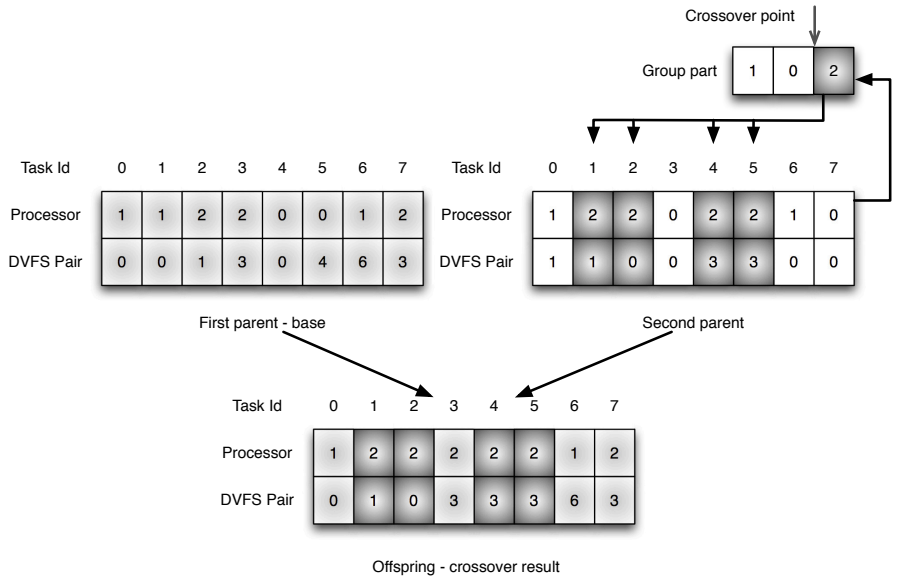


Figure 4: Example of crossover application. The child on the bottom is a result of copying subset of tasks assigned to processor number 2 (dark grey) from second parent into the base created by copying the values of first parent (light grey).

tribution. In this work, both types of mutation are independent and occur with the same probability.

Sample applications of mutation are presented in Figure 5. Mutation Type 1 changes the processor allocation and then selects a new DVFS pair: mutation occurring on position 4 modifies task allocation from processor 0 to 1 and from voltage level 0 to 6. Mutation Type 2 changes only DVFS pair selection, from

0 to 4. Note that the available range of DVFS pairs varies among processors.

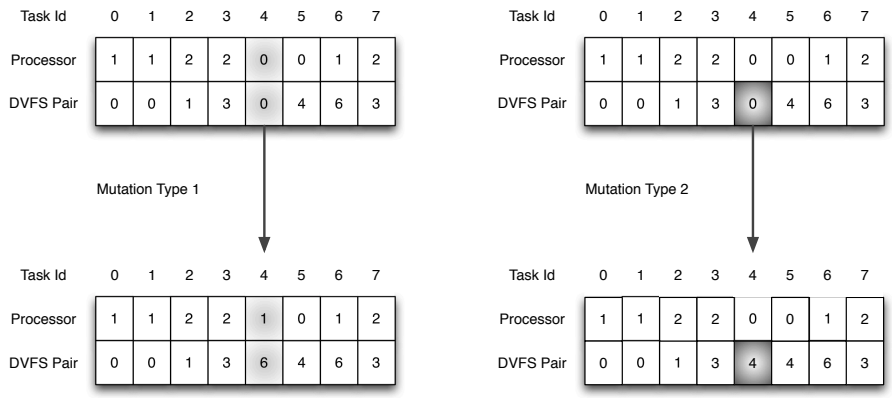


Figure 5: Examples of mutations. On the left hand side, the mutation which changes both assigned processor and DVFS pair. On the right hand side, the mutation which changes only the DVFS pair.

4.4. Evaluation

The first objective, makespan, is determined using a heuristic that translates a chromosome to a schedule. This heuristic is used in every generation for each solution so its performance is critical for the efficiency of the whole algorithm. To provide an effective answer to this problem, a greedy list heuristic, which uses insertion technique, was chosen. Insertion technique has the potential of compacting the length of the schedule, as nodes might be scheduled earlier if there is an available time period between two tasks already scheduled on a machine, where the machine runs idle. Such heuristic needs a list of tasks based on some priority. In this study, *bottom level* (b-level) [27] is used to determine tasks priorities. The b-level is defined for a task as the length of the longest path from the beginning of the task to the bottom of the graph.

All tasks are sorted according to their b-level values in non increasing order. The tasks with the highest value of this indicator are scheduled as the first ones. The priority based on b-level can be then applied on each of the processors without violating the precedence constraint [23, 26].

After the calculation of b-level and sorting the tasks, the heuristic selects a task from the beginning of the list and checks when it is possible to schedule it on the assigned processor without violating any of the constraints, and assigns the task in the first available time slot.

The energy consumed by a system is defined as E_t by Equation 2 and calculated using the schedule created by a heuristic during calculation of the makespan.

Additionally, the algorithms use also a slack reclamation technique to reduce energy consumption without increasing the makespan [28]. The slack of a task

is the time slot between the task end and the following task start. In the evaluation function, each slack is used to combine the DVFS pair selected by a MOEA with the lowest possible DVFS pair of the assigned processor to reduce the total consumed energy [6]. Finally, the minimum DVFS pair of the processor is assigned to all remaining idle time slots.

5. Experimentation

This section contains the experimental study of the algorithms and the problem. Section 5.1 describes the choice of instances for the study. Section 5.2 presents the validity of the approach proven by a set of experimental tests. Section 5.3 contains results obtained by simulations performed over the set of instances, which are subsequently discussed in Section 5.5.

5.1. Problem Instances

The diversified benchmark used in this work includes real applications DAGs, structured synthetic graphs, and random graphs. The total number of various instances used throughout the study is 2100. Instances can be described by their size: number of tasks and edges. The ratio between tasks and edges (Edge Task Ratio or ETR) gives information on the average node degree that determines the possibilities of parallelization and the number of required communications. Communication to Computation Ratio (CCR) is another metric that describes a DAG, and it is computed as the division of the average communication cost ($\overline{c}$) by the average computation cost ($\overline{p}$) on a target system, as defined by Equation 3. All instances were generated with five different CCRs: 0.1, 0.5, 1, 5 and 10 to present a wide range of possible applications, from computation-intensive (CCR = 0.1) to the communication-intensive (CCR = 10) ones. The final parameter of an instance is the processors heterogeneity values [29], which value β is used to generate the computational weight of a task p_{il}. The weight is generated using a uniform distribution around initially homogenous value p_i, defined as $\mathcal{U}(p_i \times 1 - \frac{\beta}{2}); p_i \times (1 + \frac{\beta}{2}))$. For each possible configuration, all values from the set $\{0.1, 0.25, 0.5, 0.75, 1.0\}$ were used to test a range from quasi-homogenous (heterogeneity = 0.1) to fully heterogenous (heterogeneity = 1.0) systems.

The randomization procedure used to introduce heterogeneity is composed of the following steps: first, if the homogenous weights are not included in the input graph, they are generated using a distribution $\mathcal{U}(0; 2 \times r)$, where r, in this work, is a random value from the distribution $\mathcal{U}(1; 30)$. Then, the homogenous weights are altered using the heterogeneity parameter. Finally, the edge weights are generated using the distribution $\mathcal{U}(0; 2 \times \overline{p} \times CCR)$, which ensures that the generated instance has required CCR.

$$CCR = \overline{c}/\overline{p}. \tag{3}$$

To present a wide range of possible platforms, systems containing from 8 to 128 processors were used. The generated systems were using five distinct DVFS settings presented in Table 2 and cyclically assigned to the processors. The

basic information about the size and structure of these instances is presented in Table 3.

Table 2: Processors DVFS settings used for simulations [6].

Level	Type 1		Type 2		Type 3		Type 4		Type 5	
	V_k	R_s (%)	V_k	R_s (%)	V_k	R_s (%)	V_k	R_s (%)	V_k	R_s (%)
0	1.75	100	1.50	100	2.20	100	1.95	100	1.60	100
1	1.40	80	1.40	90	1.90	85	1.60	90	1.30	85
2	1.20	60	1.30	80	1.60	65	1.30	60	1.20	60
3	0.90	40	1.20	70	1.30	50	0.90	40	0.70	40
4			1.10	60	0.90	35	0.60	20		
5			1.00	50						
6			0.90	40						

The real application graphs are represented in this study by a robot control application, a sparse matrix solver, fpppp problem from the Standard Performance Evaluation Corporation (SPEC) benchmark and part of the workflow used in Laser Interferometer Gravitational-Wave Observatory (LIGO). The instances from these set come from the Standard Task Graph (STG) homogenous set[2]. The aforementioned randomization procedure was executed on the task execution times of these instances to make them heterogeneous.

Tested system sizes were 8, 16, and 32 processors. For each of the applications, 75 instances were generated (3 processor number values, 5 heterogeneity values, 5 CCR values), which sums up to 225 real instances.

Table 3: Instance types: tasks and edges numbers, and Edge Task Ratio.

Type	Tasks	Edges	ETR
LIGO	76	132	1.73
Robot	88	131	1.48
Sparse	96	67	0.69
fpppp	334	1145	3.42

Type	Tasks	avg. ETR
GE	25–403	1.76
Laplace	25–625	1.83
Cholesky	25–625	2.60
Winkler	20–590	1.51

For the structured synthetic graphs, a larger set including 8, 16, 32, 64, and 128 processors in a system was used. The application structures are generated based on dimensions of input data matrix which are incrementally increased to produce different applications with various sizes. The sizes of instances were possible values from 25 up to 403 (Gaussian Elimination - GE) or 625 (Cholesky and Laplace). As a result, 21 task number values were used for Cholesky and Laplace and 22 values for the GE. Therefore there are 525 Cholesky and Laplace instances, and 550 GE instances.

Finally, Winkler graphs represent random graph structures of multidimensional orders [30]. Systems with 16 and 128 processors are investigated. Each graph is generated using 2-dimensional orders with. For each of n tasks a node is randomly generated in the $[0; 1]\times[0; 1]$ square. Two points are connected by an edge if both dimensions of the end point of the edge are greater than the

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

corresponding dimensions of the start point. Winkler graphs are therefore fully randomized. The instance task number covers a range from 20 to 590 with an instance every step of 30 tasks. Therefore there are 20 tasks numbers with 2 processors numbers and five CCR values which give the number of 200 instances.

5.2. Algorithms testing and configuration

This section describes how the initial experiments and algorithms configuration were conducted. It includes a presentation of the performance metrics, a convergence test and a validation of performance of the proposed recombination.

5.2.1. Performance metrics

Three independent quality indicators are selected for this study:

1. Unary Additive Epsilon ($I^1_{\varepsilon+}$,Epsilon) [31] presents the degree of convergence of the approximated set to the Pareto front. It returns the smallest distance needed for every point of the approximated set to be translated so that it dominates the Pareto front.
2. Inverted Generational Distance (IGD) [32] summarizes how close are the points in the approximated set to the closest points of the Pareto front. In contrary to Epsilon, it aggregates the results of all points. When IGD is equal to 0, it means that all points from approximated set belongs to the Pareto Front.
3. Spread [33] gives information about the distribution of the solutions along the approximated front. When it is equal to 0, it means that solutions are ideally distributed along the front.

The reason for this choice is to provide the metrics for convergence (Epsilon) as well as for distribution (Spread). The third metric (IGD) presents the aggregated results which involve convergence as well as distribution. Usage of these metrics allows to statistically analyze the results obtained from many independent runs. In order to apply these indicators, the optimal Pareto front is required. However, it is not known for the studied instances. Therefore, we build a pseudo-optimal Pareto front by aggregating the best non-dominated solutions found by all algorithms in all independent runs into one single front, for every problem instance. This pseudo-optimal Pareto front will be used to apply the indicators.

5.2.2. Convergence test

The first proof of concept for the proposed MOEAs is the convergence check. It was performed on the 36 instances from the set of real applications, which are selected to be diversified and cover different specifics of various implementations.

The convergence test was prepared in two stages. First, all the algorithms were run for 100,000 evaluations. The initial population was initialized by random values for processors assignment and the highest DVFS pairs (coded as 0). This choice is motivated by the need of finding a good assignment first and by the applied runtime slack reclamation (Section 4.4), which is able to

exploit existing slack. The probabilities of crossover and mutation were set to $p_{crossover} = 0.9$, $p_{mutation} = 1/t$ (the probability that mutation will happen for a single position in a chromosome, equal for the two mutation operators), where t is number of tasks. The selected value of the mutation probability was successfully used for combinatorial optimization problems [34, 35, 36] and it is set to apply on average a single mutation of each type in a solution, which introduces variability but is not destructive. The population and archive size were set to 100. Each experiment was independently executed 50 times. The proposed grouping crossover was used for all simulations. In the second step, the same simulations as in the previous tests were run.

Epsilon and IGD quality indicators values depend on the convergence, so they are used in this study. After each iteration, values of quality indicators of the current population are calculated using the previously prepared approximated Pareto fronts. Then, the values of 50 independent runs are averaged and plotted. The obtained convergence plots are the results of this process. Representative convergence plots are presented on Figure 6. Each generation corresponds to an evaluation of each individual in the whole population.

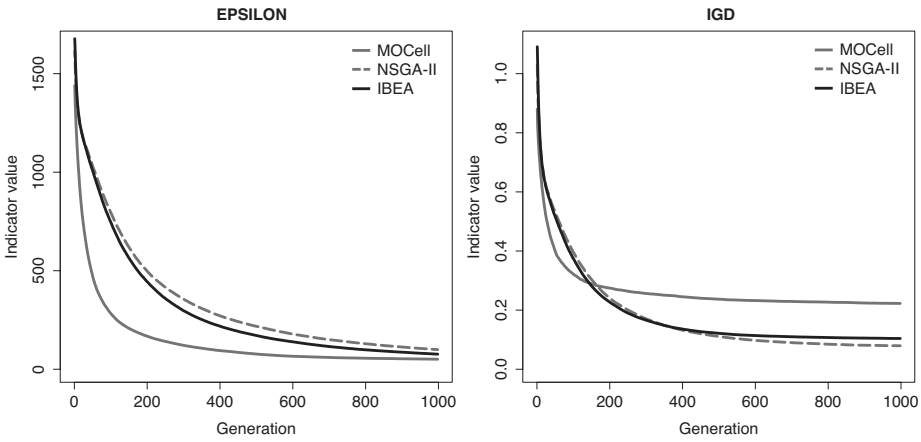


Figure 6: Convergence tests results for the fpppp instance with $m = 8$, $t = 334$, task heterogeneity= 1, $CCR = 10$.

The results of the convergence test prove that the algorithms converge in all tested cases. Both Epsilon and IGD converges to a stable value. Moreover, they present satisfactory convergence within 25,000 evaluations. The number of evaluations was used together with the settings of algorithms as presented above and implemented in jMetal. To assure strong statistical evidence, every simulation is executed independently 50 times.

5.2.3. Performance of the Proposed Recombination

To confirm the positive impact of the grouping crossover operator on the performance of the scheduling MOEAs, a set of test runs using the Real applications benchmark was conducted. Widely used two point crossover was chosen

for the comparison. As a result, each instance was run using six different configurations of MOEA and crossover (three MOEAs and two crossover operators). Values of Epsilon, IGD and Spread quality indicators are the basis for the presented comparison. Finally, the scores of the configurations were given as an input to the Friedman test.

Friedman test [37] is a non-parametric statistical procedure for multiple samples. In the studied case, there are six samples, one for each configuration. Each sample consist of events, which are average quality indicators values. These values are collected for each instance and for each quality indicator. For each event, the values are ranked by consecutive positive integer numbers: the one with highest value has rank 1, the solution with second highest value has rank 2, etc. The returned values are used to compute Friedman's statistic, which rejected the null hypothesis that the configurations have equal median values with $p - value < 2 \cdot 10^{-10}$. The average rank values are presented in Table 4. The best algorithms have the highest rank, as all quality indicators should be minimized.

The grouping recombination is ranked higher than its two-point counterpart for each algorithm. The best investigated configuration is the combination of the grouping operator and MOCell genetic algorithm. Such conclusion is consistent with the results of previous studies for a single objective scheduling of related problem [38], which underlines another feature of grouping crossover: intrinsic communication minimization. The three grouping crossover configurations are among the four best ones. Additionally, MOCell visibly outperforms the other approaches: the two best configurations include MOCell. There is no clear order among other MOEA schemes. Finally the problem is investigated using the grouping crossover, which can result in achieving accurate results, and three MOEA schemas, to explore the problem using various approaches.

Table 4: Recombination performance: results of Friedman test.

Algorithm	Ranking
MOCell Grouping	4.48
MOCell Two-point	3.98
IBEA Grouping	3.60
NSGA-II Grouping	3.10
NSGA-II Two-point	2.93
IBEA Two-point	2.91

5.3. Results

This section presents aggregated results of performance, represented by solution number and quality indicators, for each of the algorithms on all the problem instances considered to assess its advantages and disadvantages. All plots presented further, due to the large instance set, are chosen to represent the most common trends and behaviors, if it is not stated otherwise in the text.

5.3.1. Solution number analysis

The solution number analysis reflects the problem complexity. The algorithms were able to fill the archive only for the simplest instances. The results presented in Figure 7 are aggregated values of average solution number from 50 independent runs. For all instance types NSGA-II returns the biggest number of solutions, IBEA the second biggest while MOCell the smallest one. It is different only for the fpppp instances for which ranks of MOCell and IBEA are exchanged (Figure 7d). The most influential parameter for solution number is processor number (Figure 7a), as its increase is almost inversely proportional to solution number. Similar behavior but with lower impact has CCR (Figure 7b). Regarding task number, the parameter with the largest range, solution number tends to converge to a constant value or slowly increases (Figure 7c) after reaching instance size of 200 tasks. The last investigated parameter, processors heterogeneity, has the least impact (Figure 7d) slightly decreasing the solution number with its growth.

5.3.2. Quality indicators analysis

The following quality indicators study intends to answer the question how each of the algorithms performs on average in a single run. Table 5 presents the aggregated values of the quality indicators and statistical comparison between them for the three studied MOEAs. The two symbols after the number in each cell (each of them can be ▼, △, or −) indicate statistical significance of the result regarding the results of the two other algorithms. The first symbol represents relation between the algorithm indicated in the column and the first of the other two algorithms stated in the table, the second symbol between the algorithm and the second of the other two. The statistical significance is $p - value < 0.05$ in paired Wilcoxon signed-rank test with confidence level 0.95. Symbol ▼ means significantly better performance, while △ stands for significantly worse performance. The cases where no statistical difference was found are identified with dash (−). If the result is significantly better than both other algorithms (▼ ▼), the corresponding cell has the dark grey color. If the result is better than one of the others and there is no statistical difference with the other one (▼ − or − ▼), the corresponding cell has light grey color.

Table 5: The mean quality indicators values for instance type.

Instance	Epsilon			IGD			Spread		
type	MOCell	NSGA-II	IBEA	MOCell	NSGA-II	IBEA	MOCell	NSGA-II	IBEA
LIGO	302 ▼ ▼	362 △ △	335 △ ▼	2.56 − ▼	2.96 − −	3.08 △ −	0.86 ▼ ▼	1.03 △ ▼	1.05 △ △
Robot	549 − −	584 − △	554 − ▼	8.05 ▼ ▼	8.38 △ −	8.51 △ −	0.89 ▼ ▼	1.04 △ ▼	1.05 △ △
Sparse	281 ▼ ▼	380 △ △	322 △ ▼	3.48 △ △	3.12 ▼ −	3.24 ▼ −	0.81 ▼ ▼	1.01 △ ▼	1.07 △ △
fpppp	909 ▼ ▼	1830 △ △	1426 △ ▼	0.62 − △	0.50 − ▼	0.53 ▼ △	0.84 ▼ ▼	0.97 △ ▼	1.02 △ △
GE	3184 ▼ ▼	3219 △ △	3214 △ ▼	70.55 △ ▼	70.20 ▼ −	71.78 △ −	0.93 ▼ ▼	1.01 △ ▼	1.03 △ △
Laplace	3522 △ △	3369 ▼ −	3455 ▼ −	130.32 ▼ ▼	130.82 △ ▼	133.57 △ △	0.93 ▼ ▼	1.00 △ ▼	1.02 △ △
Cholesky	1448 ▼ ▼	2069 △ △	1974 △ ▼	23.58 ▼ ▼	25.24 △ −	25.76 △ −	0.90 ▼ ▼	0.98 △ ▼	1.01 △ △
Winkler	20122 − −	16936 − −	17125 − −	94.38 △ △	83.06 ▼ ▼	86.66 ▼ △	0.95 ▼ ▼	1.00 △ ▼	1.01 △ △

The average behavior of each of the algorithms is presented in Table 5. Epsilon quality indicator has the best values for MOCell, with exception for Laplace, Robot and Winkler instance types (no statistical significance was found

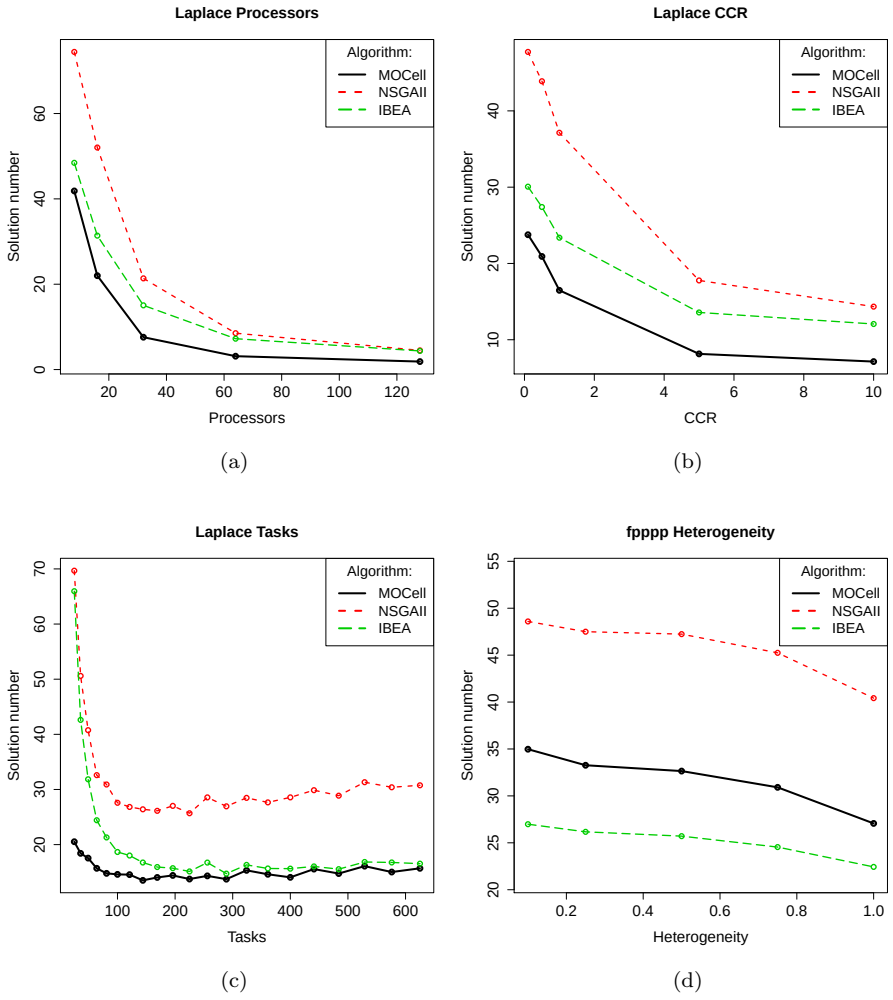


Figure 7: Aggregated solution number values.

for the two latter ones). This points out the best convergence behavior of MOCell, with respect to the other two algorithms. For IGD, MOCell is most often the statistically significantly best algorithm, but it happens only for three instance types. For LIGO instances, this algorithm is significantly better than IBEA. For the other instance types, NSGA-II is either the best algorithm for Winkler or among the best algorithms for the others. IBEA is among the best algorithms only for the Sparse instances. Results for Spread shows clear order of algorithms: MOCell provided the best results, being NSGA-II and IBEA the second and the third algorithms, respectively. This order points out the advantage of spatial structure of MOCell which increases diversification and

results in more even distribution of solutions. Additionally, MOCell is the only algorithm which achieves to be significantly better than the two others, except the NSGA-II for IGD and Winkler instances.

Three dimensional plots that show the dependency between two variables on horizontal axes and values of the quality indicator on vertical axis are used to represent the trends among the results. Analysis of such plots enables to find factors with the biggest influence on the algorithms and to formulate the rules they follow. These observations are divided into three parts, one for each quality indicator: Epsilon, IGD and Spread.

Epsilon. Epsilon values grow together with growth of processor number and CCR for all instances (Figures 8a and 8b). The influence of task number is usually not so important (Figures 8c and 8d), but for the Winkler and Cholesky instances there is a stronger positive correlation (Figure 8e). The influence of processors heterogeneity is negligible, only the fpppp instances present small negative correlation (Figure 8f).

IGD. In the sets of random and real instances IGD values grow together with growth of CCR and processor number (Figures 9a and 9b). For the synthetic graphs, presentation of results of IGD values is often suppressed by the high values obtained mostly for the high CCR and processor number, but the trend is still visible (Figure 9c). Processors heterogeneity has negligible effects: it is correlated positively with IGD for fpppp and negatively for sparse instances, which is slightly visible regarding the impact of random fluctuations (Figure 9d).

Spread. The results of analysis of spread are unexpected in terms of the opposite behavior of algorithms. For MOCell, spread grows with increasing CCR and processor number (Figures 10a and 10b); for IBEA and NSGA-II, spread decreases in the same conditions (Figures 10c and 10d). Further studies could answer if all values are asymptotically reaching value one (which could be the case concerning the results) or if this trend continues and reaches a point in which IBEA and NSGA-II are better in terms of spread than MOCell.

5.4. Comparison with other methods

To globally compare the effectiveness of the proposed optimization approach, the approximated Pareto fronts are compared with the solution given by a deterministic state-of-the-art Heterogeneous Earliest Finish Time (HEFT) [29] algorithm, which is one of the most used algorithm as a basis for comparison to evaluate the performance of new proposed scheduling algorithms. HEFT is a list-based scheduling algorithm that maintains a list of all tasks of a given graph according to their priorities. In this study we use the b-level method. The algorithm first selects a task with the highest priority for which all predecessors have been assigned. Then, a suitable machine which will result in the earliest finish time of that task is selected and the tasks is scheduled on that machine. To ensure that HEFT has competitive results, a slack reclamation post-processing step was added, which selects the slowest DVFS pair for each task that does not delay execution of other tasks.

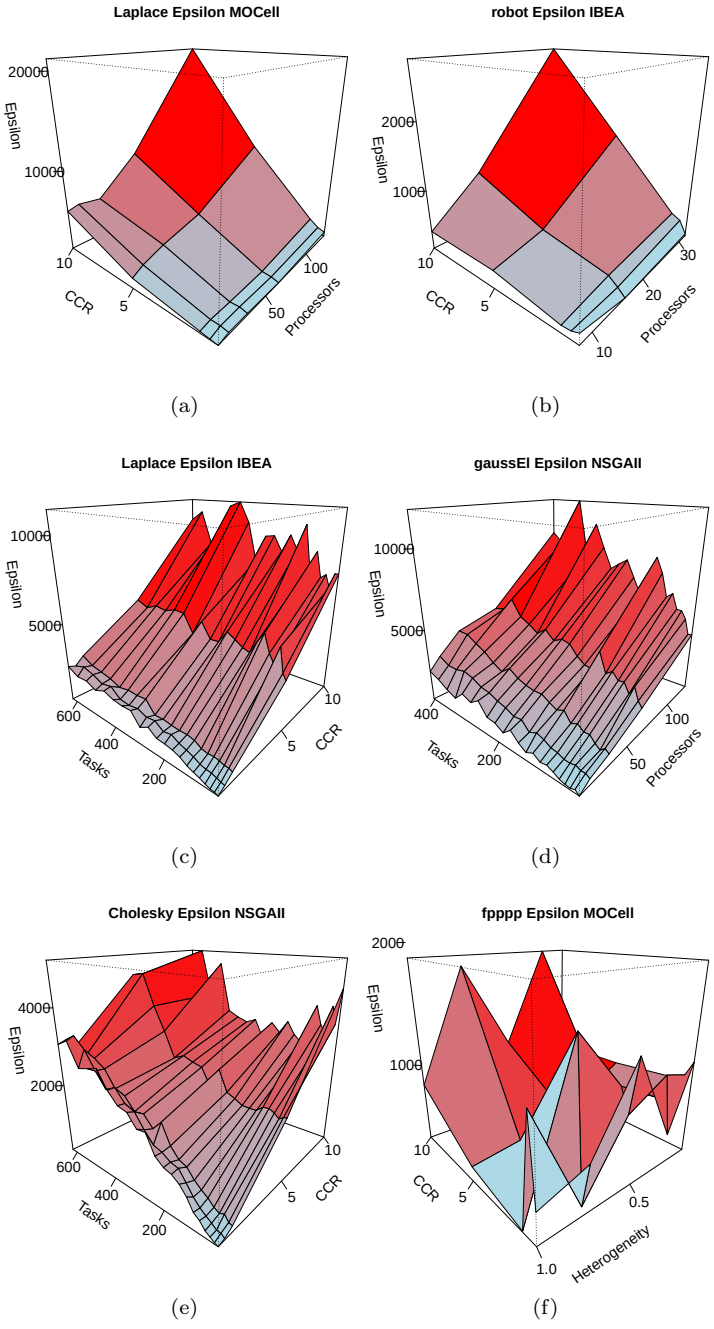


Figure 8: Three dimensional sample plots for aggregated values of Epsilon.

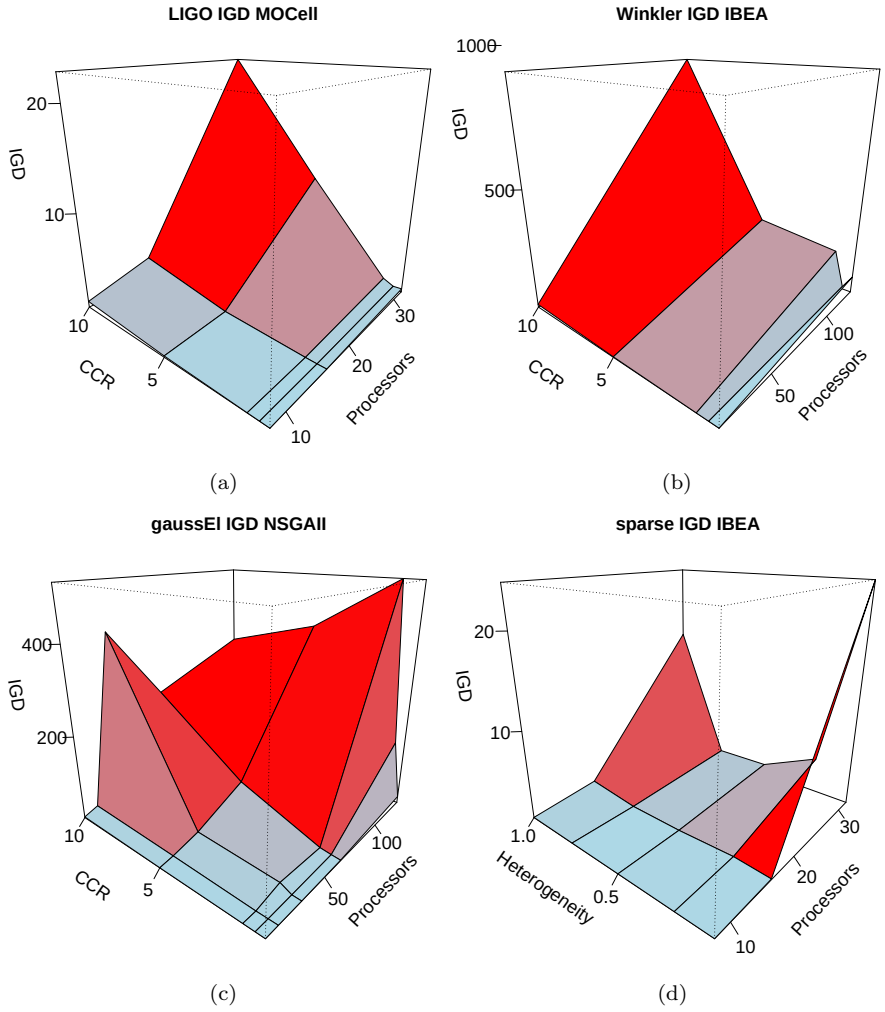


Figure 9: Three dimensional sample plots for aggregated values of IGD.

The set coverage values between the solution of HEFT and the approximated Pareto front for all considered instance types are presented in Table 6. The set coverage was selected as a method of comparison, to deal with the specific relation between the results of HEFT and MOEAs: in most of the cases they are not directly comparable, i.e. HEFT returns a single solution with the best makespan and high energy consumption, while MOEAs Pareto fronts offer a range of energy-efficient solutions with longer makespan, a representative case is presented on Figure 11. The set coverage is a ratio between number of solutions dominated in a set by the cardinality of the set and has value between 0 (when none of the point from the set is dominated) and 1 (when all points from the set are dominated).

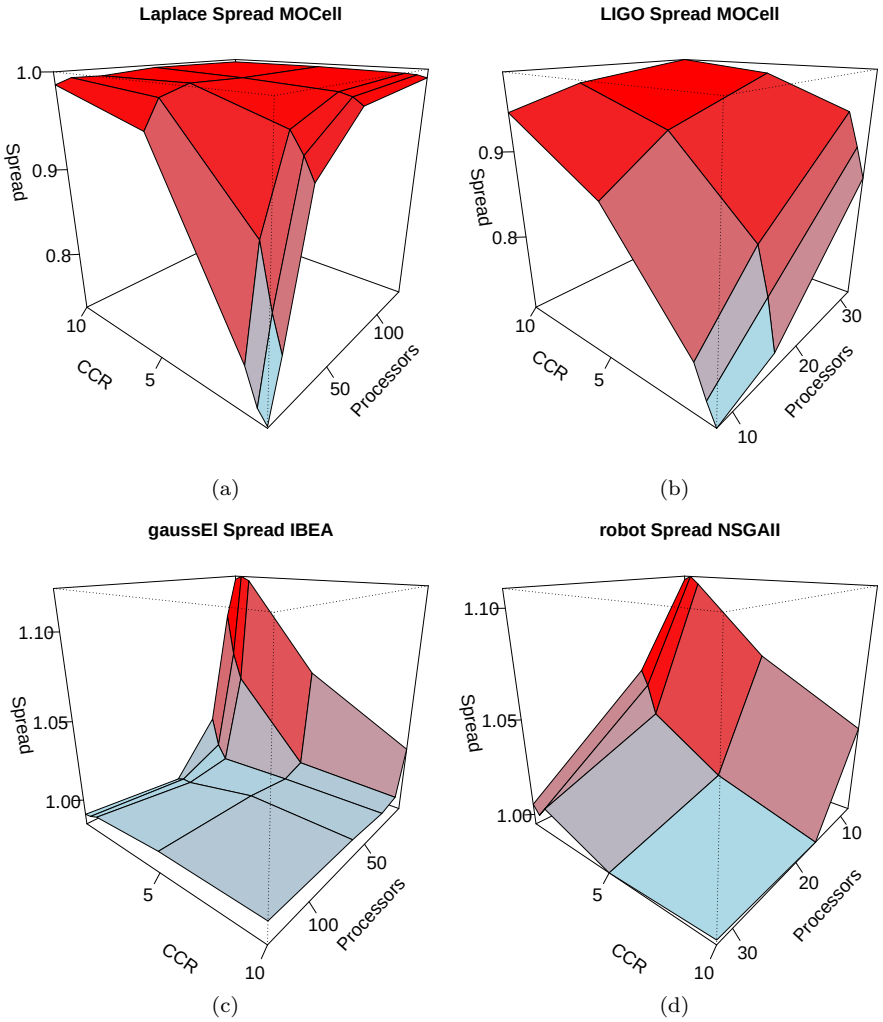


Figure 10: Three dimensional sample plots for aggregated values of Spread.

For the LIGO, Robot and Sparse types of instances MOEAs set coverage over HEFT is much bigger than the opposite. Results for fpppp instance type are incomparable: the set coverage is close to zero in both cases. Finally, for the set of synthetic and randomized applications, obtained Pareto fronts are further from the real Pareto fronts, which is supported by the high values of set coverage of HEFT over MOEAs. However, there are cases when the solutions returned by MOEAs can dominate the solution returned by HEFT. For the three first real applications the coverage results are similar: MOEAs slightly increase superiority over HEFT together with growing tasks number and decreasing ETR. Contrary to that, for the synthetic instances increasing ETR

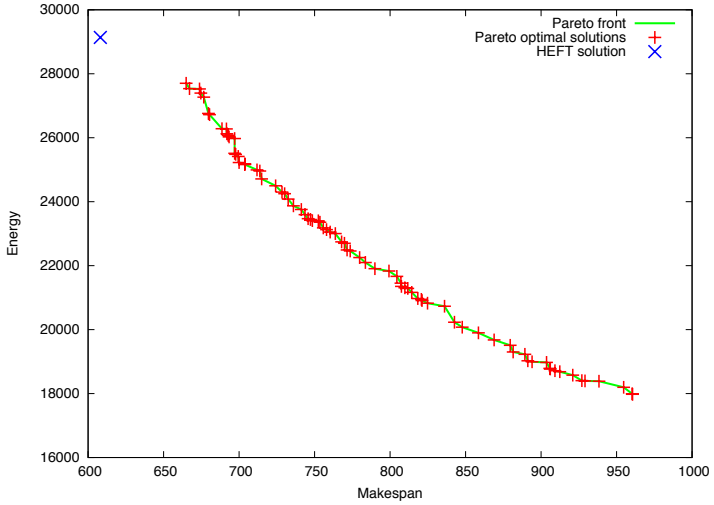


Figure 11: Sample Pareto front.

Table 6: The mean set coverage for instance type.

Instance	MOEAs/HEFT	HEFT/MOEAs
LIGO	0.187	0.013
Robot	0.200	0.120
Sparse	0.267	0.003
fpppp	0.000	0.051
GE	0.080	0.611
Laplace	0.074	0.651
Cholesky	0.103	0.465
Wickler	0.040	0.385

decrease advantage of HEFT over MOEAs.

In general, the most common behavior is that the solution of MOEAs and HEFT are incomparable. While HEFT is effective in optimizing makespan, MOEAs return sets of more energy-efficient solutions. By using multi-objective approach, it is possible to minimize energy and still provide competitive performance. Additionally, further studies on the refinement of the solutions are still possible.

5.5. Analysis of the Results

The algorithms are able to return several feasible solutions. The algorithms validation was performed on a big set of instances with different type, sizes, and characteristics. All algorithms have proven to converge to good solutions, with MOCell presenting prevalently the most effective and fast convergence. NSGA-II returns the largest number of points, while MOCell returns the smallest number. The analysis of these basic statistics does not prove clearly the final effectivity of any of these methods: the number of solutions found is satisfactory for all algorithms, and it follows the same trends which is the case also

for convergence. The analysis of quality indicators is intended to finally review the algorithms. The MOCell algorithm schema has the best average values for Epsilon and Spread and it slightly outperforms the NSGA-II average values for IGD. MOCell performed the best in the crossover comparison and quality indicators analysis for Epsilon and Spread. As the analysis of IGD also favors this algorithm and returned solution number values are similar for all algorithms, usage of MOCell is advised for this problem. Generally, it can be observed that quality indicator values increase (worsen) with the growth of processor number and CRR, and to less extent with the growth of task number. This scalability issue could be tackled by the algorithm hybridization with seeding initial population or with usage of local search.

The main outcome is not only the presented algorithms, but also the study of the problem performed by the unbiased stochastic search processes. The quality indicators study shows that processor number and CCR are the most influential characteristics of the instances regarding the quality of the obtained results. The impact of task number is lower and the impact of processors heterogeneity seems to be negligible. It is important to note that these specifics do not correspond to the computational complexity of the problem, which depends on tasks and edges number. The study shows also the fact of the importance of the instance kind on the results: depending on the instance structure, the results are very different. Analysis of number of tasks, edges or ratio between number of edges and tasks does not solve this issue solely. The algorithms present similar trends for Epsilon and IGD and two distinct behaviors for Spread.

6. Conclusions and Future Work

This study investigated the problem of scheduling precedence-constrained applications on heterogeneous, DVFS-enabled, distributed computing systems, considering simultaneously two independent objectives: schedule length and minimization of the energy consumption. Three algorithms are proposed and evaluated, based on state-of-the-art MOCell, NSGA-II and IBEA algorithms. Results show that MOEAs are able to provide accurate solutions for the addressed problem and confirm the effectivity of the method in which processor assignment and DVFS pair setting are done simultaneously, which is proven by the comparison with the HEFT algorithm. The MOEAs perform especially well in scheduling the DAGs that represents real applications. MOCell is the MOEA schema identified as the most suitable for this problem.

The results present also an exploration of the problem by testing the algorithms on the large and diversified benchmark. The difficulty of solving the energy-efficiency scheduling problem by MOEAs is dependent mainly on processor number and CCR, with minor influence of task number. These observations are supported by Wilcoxon and Friedman statistical tests performed on the values of three quality indicators: Epsilon, IGD and Spread.

There are three main directions of development, which can follow up this work. First of them is addressing the scalability issues and increasing the quality of solution for harder and bigger instances. This includes improving the

algorithmic part by such steps as seeding the initial population, adding local search step, or optimizing the algorithm runtime. Advances in these fields could facilitate addressing scheduling problems with more than two objectives: for this problem the robustness and flexibility of a schedule could be additional objectives. The last proposed research direction is refining the algorithm energy model, to include more factors such as memory energy consumption and networking system components, as well as the heat dissipation issue.

Acknowledgements

The work of P. Bouvry and J. Pecero is partly funded by INTER/CNRS/11/03 Green@Cloud. B. Dorronsoro acknowledges that the present project is partially supported by the National Research Fund, Luxembourg, and cofunded under the Marie Curie Actions of the European Commission (FP7-COFUND), under AFR contract no 4017742. M. Guzek acknowledges the support of the National Research Fund of Luxembourg (FNR) and Tri-ICT, with the AFR contract no. 1315254. Experiments presented in this paper were carried out using the HPC facility of the University of Luxembourg.

References

[1] DatacenterDynamics, Dcd industry census 2013: Data center power, `http://www.datacenterdynamics.com/focus/archive/2014/01/dcd-industry-census-2013-data-center-power` (January 2014).

[2] A. Beloglazov, R. Buyya, Y. C. Lee, A. Y. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, Advances in Computers 82 (2011) 47–111.

[3] L. Minas, B. Ellison, Energy Efficiency for Information Technology: How to Reduce Power Consumption in Servers and Data Centers, Intel Press, USA, 2009.

[4] G. Valentini, W. Lassonde, S. Khan, N. Min-Allah, S. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. Pecero, D. Kliazovich, P. Bouvry, An overview of energy efficiency techniques in cluster computing systems, Cluster Computing 16 (2013) 3–15.

[5] D. C. Snowdon, E. L. Sueur, S. M. Petters, G. Heiser, Koala: a platform for os-level power management, in: EuroSys, 2009, pp. 289–302.

[6] Y.-C. Lee, A.-Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE T Parall Distr 22 (8) (2011) 1374 –1381.

[7] M. R. Garey, D. S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.

[8] D. Zhu, D. Moss, R. Melhem, Power-aware scheduling for and/or graphs in real-time systems, IEEE T Parall Distr 15 (2004) 849–864.

[9] Y. Zhang, X. S. Hu, D. Z. Chen, Task scheduling and voltage selection for energy minimization, in: Procs. of the 39th annual Design Automation Conference, DAC '02, ACM, New York, NY, USA, 2002, pp. 183–188.

[10] G. Aupy, A. Benoit, Y. Robert, Energy-aware scheduling under reliability and makespan constraints, in: High Performance Computing (HiPC), 2012 19th International Conference on, Pune, India, 2012, pp. 1–10.

[11] L. Wang, S. Khan, D. Chen, J. Koodziej, R. Ranjan, C.-Z. Xu, A. Zomaya, Energy-aware parallel task scheduling in a cluster, Future Generation Computer Systems 29 (7) (2013) 1661–1670.

[12] S. Baskiyar, K. Palli, Low power scheduling of dags to minimize finish times, in: High Performance Computing - HiPC 2006, Vol. 4297 of LNCS, Springer Berlin / Heidelberg, 2006, pp. 353–362.

[13] S. Baskiyar, R. Abdel-Kader, Energy aware dag scheduling on heterogeneous systems, Cluster Comput 13 (2010) 373–383.

[14] N. B. Rizvandi, J. Taheri, A. Y. Zomaya, Y. C. Lee, Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms, in: Procs of the 10th IEEE/ACM CCGRID '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 388–397.

[15] V. Shekar, B. Izadi, Energy aware scheduling for dag structured applications on heterogeneous and dvs enabled processors, in: GREENCOMP '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 495–502.

[16] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E. G. Talbi, A. Y. Zomaya, D. Tuyttens, A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, J Parallel Distr Com 71 (2011) 1497–1508.

[17] J. E. Pecero, P. Bouvry, H. J. F. Huacuja, S. U. Khan, A multi-objective grasp algorithm for joint optimization of energy consumption and schedule length of precedence-constrained applications, in: DASC, IEEE, 2011, pp. 510–517.

[18] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, IEEE T Evolut Comput 6 (2) (2002) 182–197.

[19] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, E. Alba, Design issues in a multiobjective cellular genetic algorithm, in: EMO, 2006, pp. 126–140.

[20] E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search, in: PPSN, 2004, pp. 832–842.

[21] E. Alba, B. Dorronsoro, Cellular Genetic Algorithms, Vol. 42 of Operations Research/Computer Science Interfaces, Springer-Verlag Heidelberg, 2008.

[22] J. Durillo, A. Nebro, E. Alba, The jMetal framework for multi-objective optimization: Design and architecture, in: CEC 2010, Barcelona, Spain, 2010, pp. 4138–4325.

[23] O. Sinnen, Task Scheduling for Parallel Systems, John Wiley & Sons, Hoboken, NJ, USA, 2007.

[24] M. Guzek, C. O. Diaz, J. E. Pecero, P. Bouvry, A. Zomaya, Impact of voltage levels number for energy-aware bi-objective dag scheduling for multiprocessors systems, in: Advances in Information Technology, Vol. 344 of CCIS, Springer, Berlin, Heidelberg, 2012, pp. 70–80.

[25] E. Falkenauer, Genetic Algorithms and Grouping Problems, John Wiley & Sons, Inc., England, 1997.

[26] J. E. Pecero, D. Trystram, A. Y. Zomaya, A new genetic algorithm for scheduling for large communication delays, Euro-Par '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 241–252.

[27] I. Ahmad, Y.-K. Kwok, M.-Y. Wu, Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors, in: Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second Int. Symposium on, Beijing, 1996, pp. 207–213.

[28] J. E. Pecero, B. Dorronsoro, M. Guzek, P. Bouvry, Memetic algorithms for energy-aware computation and communications optimization in computing clusters, in: I. Ahmad, S. Ranka (Eds.), Handbook energy-aware and green computing, Chapman and Hall/CRC Press, 2012, pp. 443–473.

[29] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE T Parall Distr 13 (3) (2002) 260 –274.

[30] P. Winkler, Random orders, Order 1 (1985) 317–331.

[31] J. Knowles, L. Thiele, E. Zitzler, A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers, TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich (Feb. 2006).

[32] D. A. Van Veldhuizen, Multiobjective evolutionary algorithms: classifications, analyses, and new innovations, Ph.D. thesis, Wright Patterson AFB, OH, USA, adviser-Lamont, Gary B. (1999).

[33] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, A. Beham, Abyss: Adapting scatter search to multiobjective optimization, IEEE T Evolut Comput 12 (4) (2008) 439–457.

[34] K. Deb, et al., Multi-objective optimization using evolutionary algorithms, Vol. 2012, John Wiley & Sons Chichester, 2001.

[35] C. R. Reeves, J. E. Rowe, Genetic algorithms: principles and perspectives: a guide to GA theory, Vol. 20, Springer, 2003.

[36] E. A. Torres, S. Khuri, Applying evolutionary algorithms to combinatorial optimization problems, in: Computational Science-ICCS 2001, Springer, 2001, pp. 689–698.

[37] M. Friedman, The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance, Journal of the American Statistical Association 32 (200) (1937) 675–701.

[38] M. Guzek, J. E. Pecero, B. Dorronsoro, P. Bouvry, S. U. Khan, A cellular genetic algorithm for scheduling applications and energy-aware communication optimization, in: HPCS'10, Caen, France, 2010, pp. 241–248.