A Parallel Bi-objective Hybrid Metaheuristic for Energy-aware Scheduling for Cloud Computing Systems

Mohand Mezmaz, Nouredine Melab, Yacine Kessaci, Young Choon Lee, El-Ghazali Talbi, Albert Zomaya, Daniel Tuyttens

A Parallel Bi-objective Hybrid Metaheuristic for Energy-aware Scheduling for Cloud Computing Systems

M. Mezmaz[a], N. Melab[b], Y. Kessaci[b], Y. C. Lee[c], E-G. Talbi[b,d], A. Y. Zomaya[c], D. Tuyttens[a]

[a] *Mathematics and Operational Research Department (MathRO), University of Mons, Belgium*
Email: {Mohand.Mezmaz,Daniel.Tuyttens}@umons.ac.be
[b] *National Institute for Research in Computer Science and Control (INRIA), CNRS/LIFL, Université de Lille1, France*
Email: {Nouredine.Melab,Yacine.Kessaci,El-Ghazali.Talbi}@lifl.fr
[c] *Centre for Distributed and High Performance Computing, The University of Sydney, Australia*
Email: {yclee,zomaya}@it.usyd.edu.au
[d] *King Saud University, Riyadh, Saudi Arabia*

Abstract

In this paper, we investigate the problem of scheduling precedence-constrained parallel applications on heterogeneous computing systems (HCSs) like cloud computing infrastructures. This kind of applications was studied and used in many research works. Most of these works propose algorithms to minimize the completion time (makespan) without paying much attention to energy consumption.

We propose a new parallel bi-objective hybrid genetic algorithm that takes into account, not only makespan, but also energy consumption. We particularly focus on the island parallel model and the multi-start parallel model. Our new method is based on dynamic voltage scaling (DVS) to minimize energy consumption.

In terms of energy consumption, the obtained results show that our approach outperforms previous scheduling methods by a significant margin. In terms of completion time, the obtained schedules are also shorter than those of other algorithms. Furthermore, our study demonstrates the potential of DVS.

Keywords: Energy-aware Scheduling; Cloud Computing; Metaheuristics;

Hybridization; Parallelization; Genetic algorithm; Precedence-constrained parallel applications

1. Introduction

Precedence-constrained parallel applications are one of the most typical application model used in scientific and engineering fields. Such applications can be deployed on homogeneous or heterogeneous systems (HCSs) like cloud computing infrastructures.

Cloud computing is a simple concept that has emerged from heterogeneous distributed computing, grid computing, utility computing, and autonomic computing. In cloud computing, end-users do not own any part of the infrastructure. The end-users simply use the services available through the cloud computing paradigm and pay for the used services. The cloud computing paradigm can offer any conceivable form of services, such as computational resources for high performance computing applications, web services, social networking, and telecommunications services. Several criteria determine the quality of the provided service, the production cost of this service, and therefore the price paid by the end-user. The duration of this service (makespan) and the consumed energy are among of these criteria. The idea is to provide end-users a more flexible service that takes into account the duration of the service and the consumed energy. End-users could then find the right compromise between these two conflicting objectives to solve their precedence-constrained parallel applications.

The problem of finding the right compromise between the resolution time and the energy consumed of a precedence-constrained parallel application is a bi-objective optimization problem. The solution to this problem is a set of Pareto points. Pareto solutions are those for which improvement in one objective can only occur with the worsening of at least one other objective. Thus, instead of a unique solution to the problem, the solution to a bi-objective problem is a (possibly infinite) set of Pareto points. To the best of our knowledge, there is no research published in the literature to solve the above problem with a Pareto approach.

However, many works have focused on precedence-constrained parallel applications (e.g., [7], [31], [25] and [19]). Most of these works propose algorithms to minimize the makespan. Only recently that some works are interested in minimizing the energy consumption (e.g., [13], [2], [29], [30], [10] and [23]).

Using green IT techniques can significantly reduce an organization's and ultimately a country's carbon footprint. The UN bought 461 tons of carbon offsets to ensure that the September 2009 Summit on Climate Change in New York was carbon neutral. According to [3], if green IT techniques were implemented that reduced the energy use of 10,000 computers in West Virginia by 50%, the deployment would produce 33 times less carbon in one year than that produced by the summit. A recent study on power consumption by servers [16] shows that, in 2005, the power used by servers represented about 0.6% of total U.S. electricity consumption. That number grows to 1.2% when cooling and auxiliary infrastructures are included. In the same year, the aggregate electricity bill for operating those servers and associated infrastructure was about $2.7 billions and $7.2 billions for the U.S. and the world, respectively. The total electricity use for servers doubled over the period 2000 to 2005 in worldwide. The number of transistors integrated into today's Intel Itanium 2 processor reaches nearly 1 billion. If this rate continues, the heat (per square centimeter) produced by future Intel processors would exceed that of the surface of the sun [15]. This implies the possibility of worsening system reliability, eventually resulting in poor system performance.

Due to the importance of energy consumption, various techniques including dynamic voltage scaling (DVS), resource hibernation, and memory optimizations have been investigated and developed [26]. DVS among these has been proven to be a very promising technique with its demonstrated capability for energy savings (e.g., [2], [29] and [10]). For this reason, we adopt this technique and it is of particular interest to this study. DVS enables processors to dynamically adjust voltage supply levels (VSLs) aiming to reduce power consumption. However, this reduction is achieved at the expense of sacrificing clock frequencies.

In this paper, we investigate the energy issue in task scheduling particularly on HCSs like cloud computing systems. We propose a new parallel bi-objective hybrid genetic algorithm that takes into account, not only makespan, but also energy consumption. Our new approach is a hybrid between a multi-objective parallel genetic algorithm and energy-conscious scheduling heuristic (ECS) [20]. The results clearly demonstrate the superior performance of ECS over the other algorithms like DBUS [1] and HEFT [25]. Genetic algorithms make it possible to explore a great range of potential solutions to a problem. The exploration capability of the genetic algorithm and the intensification power of ECS are complementary. A skillful combination of a metaheuristic with concepts originating from other types of algorithms lead to more efficient behavior.

Our algorithm is effective as it profits from the exploration power of the genetic algorithm, the intensification capability of ECS, the cooperative approach of the island model, and the parallelism of the multi-start model. The island model and the hybridization improve the quality of the obtained results. The multi-start model reduces the running time of a resolution. Furthermore, one of the major interests of our approach is to give the end-user a set of Pareto solutions to choose according to the desired quality of service, in particular the completion time, and the cost of the service in terms of energy and consequently in terms of price willing to pay. The proposed method can easily be applied to loosely coupled HCSs (e.g., cloud computing systems) using advance reservation and various sets of frequency–voltage pairs.

Our new approach is evaluated with the Fast Fourier Transformation task graph which is a real-world application. Experiments show that (1) the hybridization improves on average the best known results obtained in the literature (by **47.5%** for the energy consumption and **12%** for the completion time), (2) the island model significantly improves the results obtained using only the hybridization, and (3) the multi-start model accelerates our approach with an average speedup of **13** using 21 cores.

The remainder of the paper is organized as follows. Section 2 presents the application, system, energy and scheduling models used in this paper. Section 3 describes the related work. Our algorithm is presented in Section 4. The results of our comparative experimental study are discussed in Section 5. The conclusion is drawn in Section 6. The paper ends with an appendix which describes our approaches using pseudo-code.

2. Problem modeling

In this section, we describe the system, application, energy and scheduling models used in our study.

2.1. Cloud computing model

A cloud computing system is a set of resources designed to be allocated ad hoc to run applications. In our model, the cloud is assumed to be hosted in a data center which is composed by heterogeneous machines. This data center provides a set of services hosted on thousands of high-end computing servers.

The need in terms of services of an application can be modeled by a task graph. In this graph, an edge between two tasks represents an inter-service communication.

The cloud computing system used in this work consists of a set P of p heterogeneous processors/machines. Each processor $p_j \in P$ is DVS-enabled; in other words, it can operate with different VSLs (i.e., different clock frequencies). For each processor $p_j \in P$, a set V_j of v VSLs is random and uniformly distributed among three different sets of VSLs (Table 1). Since clock frequency transition overheads take a negligible amount of time (e.g., $10\mu s$- $150\mu s$ [11], [22]), these overheads are not considered in our study. The inter-processor communications are assumed to perform with the same speed on all links without contentions. It is also assumed that a message can be transmitted from one processor to another while a task is being executed on the recipient processor which is possible in many systems.

Table 1: Voltage-relative speed pairs

	Pair 1		Pair 2		Pair 3	
Level	Voltage (v_k)	Relative speed (%)	Voltage (v_k)	Relative speed (%)	Voltage (v_k)	Relative speed (%)
0	1.5	100	2.2	100	1.75	100
1	1.4	90	1.9	85	1.4	80
2	1.3	80	1.6	65	1.2	60
3	1.2	70	1.3	50	0.9	40
4	1.1	60	1.0	35		
5	1.0	50				
6	0.9	40				

2.2. Application model

Parallel programs can be generally represented by a directed acyclic graph (DAG). A DAG, $G = (N, E)$, consists of a set N of n nodes and a set E of e edges. A DAG is also called a task graph or macro-dataflow graph. In general, the nodes represent tasks partitioned from an application; the edges represent precedence constraints. An edge $(i, j) \in E$ between task n_i and task n_j also represents inter-task communication. A task with no predecessors is called an entry task, n_{entry}, whereas an exit task, n_{exit}, is one that does not have any successors. Among the predecessors of a task n_i, the predecessor which completes the communication at the latest time is called the most influential parent (MIP) of the task denoted as $MIP(n_i)$. The longest path of a task graph is the critical path.

The weight on a task n_i denoted as w_i represents the computation cost of the task. In addition, the computation cost of the task on a processor p_j, is denoted as $w_{i,j}$ and its average computation cost is denoted as $\bar{w}_i$.

The weight on an edge, denoted as $c_{i,j}$ represents the communication cost between two tasks, n_i and n_j. However, a communication cost is only required when two tasks are assigned to different processors. In other words, the communication cost when tasks are assigned to the same processor is zero and thus can be ignored.

The earliest start time of, and the earliest finish time of, a task n_i on a processor p_j is defined as

$$EST(n_i, p_j) = \left\{ \begin{smallmatrix} 0 & if n_i = n_{entry} \\ EFT(MIP(n_i), p_k) + c_{MIP(n_i),i} & otherwise \end{smallmatrix} \right.$$

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j}$$

Note that the actual start and finish times of a task n_i on a processor p_j, denoted as $AST(n_i, p_j)$ and $AFT(n_i, p_j)$ can be different from its earliest start and finish times, $EST(n_i, p_j)$ and $EFT(n_i, p_j)$, if the actual finish time of another task scheduled on the same processor is later than $EST(n_i, p_j)$.

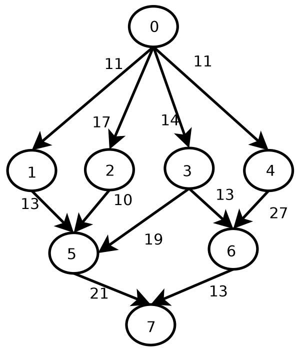

Figure 1: A simple task graph

In the case of adopting task insertion the task can be scheduled in the idle time slot between two consecutive tasks already assigned to the processor as long as no violation of precedence constraints is made. This insertion scheme would contribute in particular to increasing processor utilization for a communication intensive task graph with fine-grain tasks.

A simple task graph is shown in Figure 1 with its details in Table 2 and Table 3. The values presented in Table 3 are computed using two frequently used task prioritization methods, t-level and b-level. Note that, both computation and communication costs are averaged over all nodes and links. The t-level of a task is defined as the summation of the computation and communication costs along the longest path of the node from the entry task in the task graph. The task itself is excluded from the computation. In contrast, the b-level of a task is computed by adding the computation and communication costs along the longest path of the task from the exit task in the task graph (including the task). The b-level is used in this study.

The communication to computation ratio (CCR) is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system.

Table 2: Computation cost with VSL 0

Task	p_0	p_1	p_2
0	11	13	9
1	10	15	11
2	9	12	14
3	11	16	10
4	15	11	19
5	12	9	5
6	10	14	13
7	11	15	10

2.3. Energy model

Our energy model is derived from the power consumption model in complementary metal-oxide semiconductor (CMOS) logic circuits. The power consumption of a CMOS-based microprocessor is defined to be the summation of capacitive, short-circuit and leakage power. The capacitive power

Table 3: Task Priorities

Task	b-level	t-level
0	101.33	0.00
1	66.67	22.00
2	63.33	28.00
3	73.00	25.00
4	79.33	22.00
5	41.67	56.33
6	37.33	64.00
7	12.00	89.33

(dynamic power dissipation) is the most significant factor of the power consumption. The capacitive power (P_c) is defined as

$$P_c = ACV^2 f, \tag{1}$$

where A is the number of switches per clock cycle, C is the total capacitance load, V is the supply voltage, and f is the frequency. Equation (1) clearly indicates that the supply voltage is the dominant factor; therefore, its reduction would be most influential to lower power consumption. The energy consumption of the execution of a precedence-constrained parallel application used in this study is defined as

$$E = \sum_{i=0}^{n} ACV_i^2 f.w_i^* = \sum_{i=0}^{n} \alpha V_i^2 w_i^*,$$

where V_i is the supply voltage of the processor on which task n_i is executed, and w_i^* is the computation cost of task n_i (the amount of time taken for n_i's execution) on the scheduled processor.

2.4. Scheduling model

The task scheduling problem in this study is the process of allocating a set N of n tasks to a set P of p processors (without violating precedence constraints) that minimizes makespan with energy consumption as low as possible. The makespan is defined as $M = max\{AFT(n_{exit})\}$ after the scheduling of n tasks in a task graph G is completed. Although the minimization of makespan is crucial, tasks of a DAG in our study are not associated with deadlines as in real-time systems.

3. Related work

In this section, we present some noteworthy works in task scheduling, particularly for HCSs, and then scheduling algorithms with power/energy consciousness.

3.1. Scheduling in HCSs

Due to the NP-hard nature of the task scheduling problem in general cases [9], heuristics, in particular meta-heuristics, are the most popularly adopted scheduling approaches. List scheduling heuristics are the dominant heuristic technique. This is because empirically, list scheduling algorithms tend to produce competitive solutions with lower time complexity compared to algorithms in the other categories [17].

The HEFT algorithm [25] is highly competitive in that it generates a schedule length comparable to other scheduling algorithms, with a low time complexity ($O(nlogn+(e+n)p)$). It is a list-scheduling heuristic consisting of the two typical phases of list scheduling (i.e., task prioritization and processor selection) with task insertion.

The DBUS algorithm [1] is a duplication-based scheduling heuristic that first performs a critical path based listing for tasks and schedules them with both task duplication and insertion. The experimental results in [1] show its attractive performance, especially for communication-intensive task graphs. The time complexity of DBUS is in the order of $O(n^2p^2)$.

3.2. Scheduling with energy consciousness

Most of previous studies on scheduling with the consideration of energy consumption are conducted on homogeneous computing systems [13], [29], [30], [10], [23], [4] or single-processor systems [28]. In addition to system homogeneity, tasks are generally homogeneous and independent. Slack management/reclamation is a frequently adopted technique with DVS.

In [29], several different scheduling algorithms using the concept of slack sharing among DVS-enabled processors were proposed. The rationale behind the algorithms is to utilize idle (slack) time slots of processors lowering supply voltage (frequency/speed). This technique is known as slack reclamation. These slack time slots occur, due to earlier completion (than worst case execution time) and/or dependencies of tasks. The work in [29] has been extended in [30] with AND/OR model applications. Since the target system for both works is shared-memory multiprocessor systems, communication between dependent tasks is not considered unlike our approach.

In [10], two voltage scaling algorithms for periodic, sporadic, aperiodic tasks on a dynamic priority single-processor system are proposed. They are more practical compared with many existing DVS algorithms in that *a priori* information on incoming tasks is not assumed to be available until the tasks are actually released. This assumption does not correspond to our task model as explained.

Rountree et al. in [23] developed a system based on linear programming that exploits slack using DVS (i.e., slack reclamation). Their linear programming system aims to deliver near-optimal schedules that tightly bound optimal solutions. It incorporated allowable time delays, communication slack, and memory pressure into its scheduling. The linear programming system mainly deals with energy reduction for a given pre-generated schedule with a makespan constraint as in most existing algorithms. In our apprche, the makespan is not a constraint but an objective to optimize.

Another two scheduling algorithms for bag-of-tasks applications on clusters are proposed in [13]. Tasks in a bag-of-tasks application are typically independent and homogeneous, yet run with different input parameters/files. In [13], deadline constraints are associated with tasks for the purpose of quality control. The two algorithms differ in terms of whether processors in a given computer cluster are time-shared or space-shared. Computer clusters in this paper are composed of homogeneous DVS-enabled processors unlike our approach where processors are heterogeneous.

In [27], the authors propose a formulation of energy aware scheduling algorithm and a detailed discussion of slack time computation. This scheduling algorithm also concerns reducing voltages during the communication phases between parallel jobs. In [12], the authors study the energy-aware task allocation problem for assigning a set of tasks onto the machines of a computational grid each equipped with DVS feature. The goal is to optimize the energy consumption and response time in computational grids. Unlike our approach, [12] suppose that tasks are independent and are not subject to precedence constraints.

To the best of our knowledge, none of previous scheduling approaches explicitly addresses the energy issue with a multi-objective approach when tackling the problem of scheduling precedence-constrained parallel applications on HCSs. Therefore, the scheduling algorithms with energy consciousness presented in this section are the most closely related works to our study.

3.3. Energy-conscious scheduling heuristic

The consideration of energy consumption in task scheduling adds another layer of complexity to an already intricate problem. Unlike real-time systems, applications in our study are not deadline-constrained. Therefore, the evaluation of the quality of schedules should be measured explicitly considering both makespan and energy consumption. For this reason, energy-conscious scheduling heuristic (ECS) [20] is devised with relative superiority (RS) as a novel objective function, which takes into account these two performance criteria.

For a given ready task, its RS value on each processor is computed using the current best combination of processor and VSL (p' and v' are, respectively, the selected processor and its voltage supply level) for that task, and then the processor—from which the maximum RS value is obtained—is selected. Since each scheduling decision that ECS makes tends to be confined into a local optimum, another energy reduction technique (MCER) is incorporated into the energy reduction phase of ECS without sacrificing time complexity. It is an effective technique in lowering energy consumption, although the technique may not help schedules escape from local optima. MCER is makespan conservative in that changes it makes (to the schedule generated in the scheduling phase) are only validated if they do not increase the makespan of the schedule. For each task in a DAG, MCER considers all of the other combinations of task, host and VSL to check whether any of these combinations reduces the energy consumption of the task without increasing the current makespan.

The results clearly demonstrate the superior performance of ECS over DBUS and HEFT. Note that, in many previous studies [14], [18], HEFT has been proven to perform very competitively, and it has been frequently adopted and extended.

However, ECS returns one solution as a result, and precedence-constrained applications problem is bi-objective in nature. Section 4 presents our new parallel bi-objective approach based on hybridization between a genetic algorithm and ECS. This approach provides a set of solutions to this problem. The experiments presented in Section 5 show that our approach often gives solutions which are better than those found by ECS.

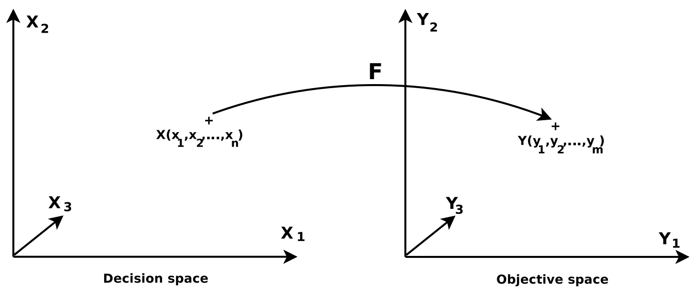

Figure 2: Illustration of a MOP

4. A parallel hybrid approach

This section starts with a brief overview on multi-objective combinatorial optimization and genetic algorithms. Then, our new parallel bi-objective hybrid approach is presented.

4.1. Multi-objective Combinatorial Optimization

A multi-objective optimization problem (MOP) consists generally in optimizing a vector of nb_{obj} objective functions $F(x) = (f_1(x), \dots,\ f_{nb_{obj}}(x))$, where x is an d-dimensional decision vector $x = (x_1, \dots, x_d)$ from some universe called *decision space*. The space the objective vector belongs to is called the *objective space*. F can be defined as a cost function from the decision space to the objective space that evaluates the quality of each solution $(x_1, \dots, x_d)$ by assigning it an objective vector $(y_1, \dots, y_{nb_{obj}})$, called the *fitness* (Figure 2). While single-objective optimization problems have a unique optimal solution, a MOP may have a set of solutions known as the *Pareto optimal set*. The image of this set in the objective space is denoted as *Pareto front*. For minimization problems, the Pareto concepts of MOPs are defined as follows (for maximization problems the definitions are similar):

- **Pareto dominance**: An objective vector y^1 dominates another objective vector y^2 if no component of y^2 is smaller than the corresponding component of y^1, and at least one component of y^2 is greater than its correspondent in y^1 i.e.:
$$\begin{cases} \forall i \in [1..nb_{obj}], y_i^1 \leq y_i^2 \\ \exists j \in [1..nb_{obj}], y_j^1 < y_j^2 \end{cases}$$

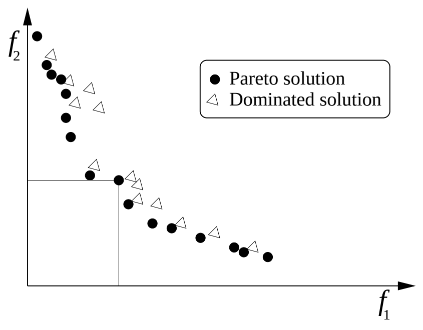

Figure 3: Example of non-dominated solutions

- **Pareto optimality:** A solution x of the decision space is Pareto optimal if there is no solution x' in the decision space for which $F(x')$ dominates $F(x)$.
- **Pareto optimal set:** For a MOP, the Pareto optimal set is the set of Pareto optimal solutions.
- **Pareto front:** For a MOP, the Pareto front is the image of the Pareto optimal set in the objective space.

Graphically, a solution x is Pareto optimal if there is no other solution x' such that the point $F(x')$ is in the dominance cone of $F(x)$. This dominance cone is the box defined by $F(x)$, its projections on the axes and the origin (Figure 3).

4.2. Genetic algorithms

Genetic Algorithms (GAs) are meta-heuristics based on the iterative application of stochastic operators on a population of candidate solutions. At each iteration, solutions are selected from the population. The selected solutions are recombined in order to generate new ones. The new solutions replace other solutions selected either randomly or according to a selection strategy.

In the Pareto-oriented multi-objective context [8], the structure of the GA remains almost the same as in the mono-objective context. However, some adaptations are required mainly for the evaluation and selection operators.

The selection process is often based on two major mechanisms: *elitism* and *sharing*. They allow respectively the convergence of the evolution process to the best Pareto front and to maintain some diversity of the potential solutions. The elitism mechanism makes use of a second population called a *Pareto archive* that stores the different non-dominated solutions generated through the generations. Such an archive is updated at each generation and used by the selection process. Indeed, the individuals on which the variation operators are applied are selected either from the Pareto archive, from the population or from both of them. The sharing operator maintains the diversity on the basis of the similarity degree of each individual compared to the others. The similarity is often defined as the Euclidean distance in the objective space.

4.3. Hybrid approach

In our approach illustrated in Figure 4, a solution (chromosome) is composed of a sequence of N genes. The i^{th} gene of a solution s is denoted s_j. Each gene is defined by a task, a processor and a voltage. These three parts of s_j are denoted respectively $t(s_j)$, $p(s_j)$ and $v(s_j)$. This means that the task $t(s_j)$ is assigned to the processor $p(s_j)$ with the voltage $v(s_j)$.

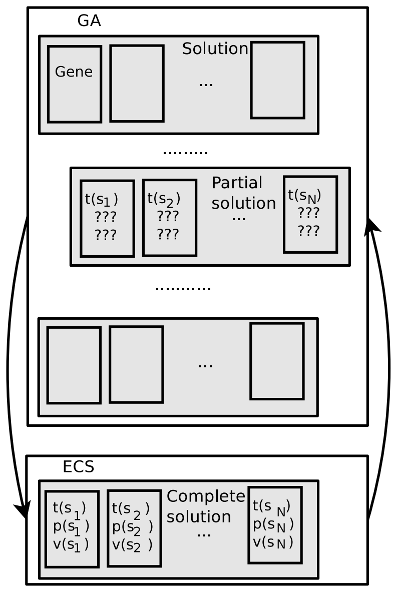

Figure 4: Our hybrid GA (GA and ECS)

The new approach we propose is based on ECS [20] which is not a population-based heuristic. ECS tries to construct in a greedy way one solution using three components.

- The first component to build the task parts of each gene of the solution.
- The second component to build the processor and voltage parts of these genes.
- And the third component to calculate the fitness of a solution in terms of energy consumption and makespan.

Unlike ECS, our approach provides a set of Pareto solutions. This approach is a hybrid between a multi-objective GA and the second component of ECS. The role of the GA is to provide good task scheduling. In other words, the GA builds task parts $t(s_1)$, $t(s_2)$, ..., $t(s_n)$ of a solution s. Therefore, the mutation and crossover operators of the GA affect only the task part of the genes of each solution.

The second component of ECS is called whenever a solution is modified by these two operators. The first role of this component is to correct the task order to take into account the precedence constraints in the task graph. Then the component completes the processor and voltage parts of the genes of the partial solutions provided by these operators. In other words, ECS builds the remaining parts $p(s_1)$, $p(s_2)$, ..., $p(s_n)$ and $v(s_1)$, $v(s_2)$, ..., $v(s_n)$ of the partial solutions provided by the mutation and crossover operators of the GA.

The evaluation (fitness) operator of the GA is called once the task, processor and voltage parts of each gene of the solution are known. The role of this operator is to calculate the energy consumption and the makespan of each solution.

The mutation operator is based on the first component of ECS. This first component returns all tasks scheduled according to their b-level values. The principle of our mutation operator is also based on the scheduling of tasks according to their b-level values. The b-level concept is explained in Section 2. It should be noted that one can choose the t-level values instead of those of b-level. First, the operator chooses randomly two integers i and j such that $1 \leq i < j \leq n$ and $b - level(t(s_i)) < b - level(t(s_j))$. Then, the operator swaps the two tasks $t(s_i)$ and $t(s_j)$ (see Figure 5).

As illustrated in Figure 6, the crossover operator uses two solutions $s1$ and $s2$ to generate two new solutions $s'1$ and $s'2$. To generate $s'1$, the operator:

- considers $s1$ as the first parent and $s2$ as the second parent.
- randomly selects two integers i and j such that $1 \leq i < j \leq n$.

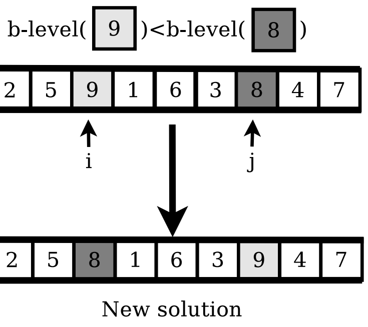

Figure 5: The mutation operator

- copies in $s'1$ all tasks of $s1$ located before i or after j. These tasks are copied according to their positions ($s'1_k = s1_k$ if $k < i$ or $k > j$).
- copies in a solution s all tasks of $s2$ that are not yet in $s'1$. Thus, the new solution s contains $(j - i + 1)$ tasks. The first task is at position 1 and the last task at the position $(j - i + 1)$.
- and finally, copies all the tasks of s to the positions of $s'1$ located between i and j ($s'1_k = s_{k-i+1}$ for all $i \leq k \leq j$).

The solution $s'2$ is generated with the same method by considering $s2$ as the first parent and $s1$ as the second parent.

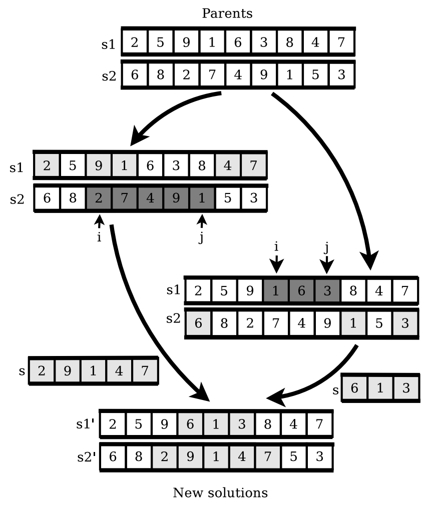

Figure 6: The crossover operator

The other elements of the GA in the new approach are conventional. Indeed, our GA randomly generates the initial population. Its selection operator is based on a tournament strategy. The algorithm stops when no new best solution is found after a fixed number of generations.

4.4. Insular approach

The island model [5] is inspired by behaviors observed in the ecological niches. In this model, several evolutionary algorithms are deployed to evolve simultaneously various populations of solutions, often called islands. As shown in Figure 7, the GAs of our hybrid approach asynchronously exchange solutions. This exchange aims at delaying the convergence of the evolutionary process and to explore more zones in the solution space. For each island, a migration operator intervenes at the end of each generation. Its role consists to decide the appropriateness of operating a migration, to select the population sender of immigrants or the receiver of emigrants, to choose the emigrating solutions, and to integrate the immigrant ones.

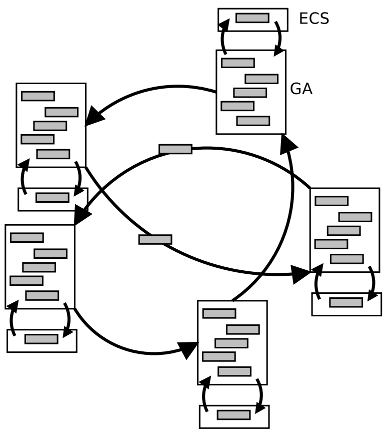

Figure 7: The cooperative island approach

4.5. Multi-start approach

Compared to the GA, ECS is more costly in CPU time. The different evaluations of ECS are independent of each other. Therefore, their parallel execution can make the approach faster. The objective of the hybrid approach is to improve the quality of solutions. The island approach also aims to obtain solutions of better quality. The goal of the parallel multi-start approach is to reduce the execution time. As shown in Figure 8, our parallelization is based on the deployment of the approach using the farmer-worker paradigm. The GA processes are farmers and ECS processes are workers.

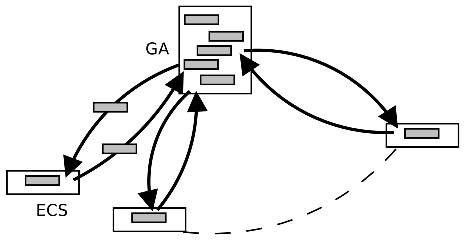

Figure 8: Illustration of the multi-start approach

5. Experiments and results

This section presents the results obtained from our comparative experimental study. The experiments aim to demonstrate and evaluate the contribution of the hybridization, the insular approach and the multi-start approach respectively compared to ECS, the hybrid approach and the insular approach.

5.1. Experimental settings

The performance of our approach was thoroughly evaluated with the Fast Fourier Transformation [6] task graph which is a real-world application. A large number of variations were made on this task graph for more comprehensive experiments. Various different characteristics of processors were also applied. Table 4 summarizes the parameters used in our experiments.

Table 4: Experimental parameters

Parameter	Value
The number of tasks	∼20 ∼40 ∼60 ∼80 ∼120
The number of processors	02 04 08 16 32 64
Processor heterogeneity	100 200 random
CCR	0.1 0.2 1.0 5.0 10.0

The new approach is experimented on 9,000 instances distributed equitably according to the number of tasks, the number of processors, the processor heterogeneity and the CCR (1/5 of instances have a number of tasks equal to ∼20, 1/5 of instances have a number of tasks equal to ∼40,..., 1/6 of instances have a number of processors equal to 2, etc.). In other words, 20 instances are used for each combination of parameters.

Our approach has been implemented using ParadisEO [21]. This software platform provides tools for the design of parallel meta-heuristics for multi-objective optimization. [24] explains how to implement a multi-objective genetic algorithm, an insular approach, and a multi-start using ParadisEO. Table 5 shows the parameters used by ParadisEO for the hybrid, insular and multi-start approaches during our experiments.

Table 5: The parameters used by ParadisEO for each approach

Parameters	Hybrid	Insular	Multi-start
Population size	20	20	20
Number of generations	1000	100	100
Crossover rate	1	1	1
Mutation rate	0.35	0.35	0.35
Migration topology		Ring	
Migration rate		Every 20 generations	
Number of migrants		5	

Experiments have been performed on a grid of three clusters. The first cluster contains 8 Opteron 244 nodes (dual-processor clocked at 1.8 GHz, 2 GB of RAM). The second contains 10 Xeon L5420 nodes (bi-quad-core processors clocked at 2.5 GHz, 16 GB of RAM). The third cluster contains 106 cores AMD opteron 248, 40 cores AMD opteron 252, 104 cores AMD opteron 285, et 368 cores Intel Xeon E5440 QC. A total of 714 cores are used. The first two clusters are located at the University of Mons in Belgium, while the third cluster is at Université de Lille1 in France.

5.2. Hybrid approach

The hybrid approach is experimented on all instances of Table 4. Each instance is solved twice. The first resolution is done with ECS, and the

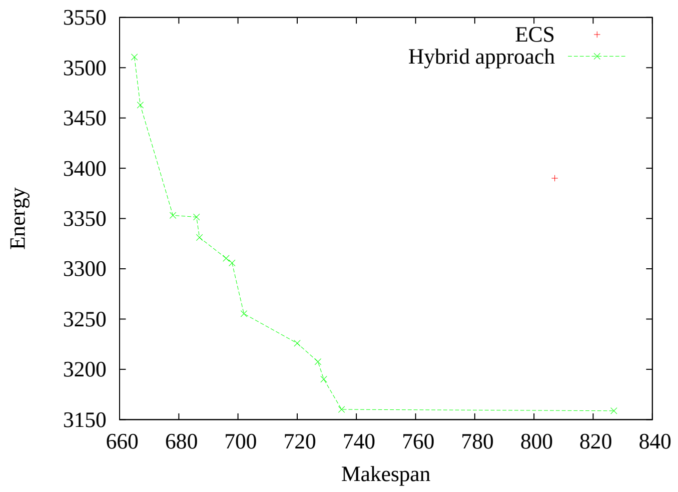

Figure 9: An example of the obtained results with the hybrid approach and ECS for the same instance

second resolution with the new approach. These experiments are launched by a script on one of the cores of our grid according to their availability.

Figure 9 gives an example of a Pareto front obtained with the hybrid approach and the solution obtained by ECS for the same instance which is the tenth instance generated with the number of tasks equal to ∼20, number of processors equal to 02, processor heterogeneity equal to 100, and CCR equal to 0.1. Experiments show that ECS finds the solution of an instance after ~ 1 second on average, while our hybrid approach requires about ~ 25 minutes on average to find the Pareto solutions of an instance. As previously mentioned, ECS is a heuristic that builds one and only one solution using a greedy strategy, and our hybrid approach is based on the hybridization between a GA and ECS. Therefore, the hybrid approach handles a population of solutions that evolves over several generations, and the second component of ECS is called and used during the construction of each solution. So it was expected that the hybrid approach uses more computing power than ECS. Our goal is not to have an approach faster than ECS but an approach that gives Pareto solutions which improve the solution of ECS. Our approach is useful, for example, for large scientific applications requiring high computing power, and for small applications which are executed many times.

Table 6 compares the Pareto solutions of the hybrid approach with the solution of ECS. The comparison is made according to the number of tasks, the number of processors, the processor heterogeneity and the CCR. The third column shows the average number of obtained Pareto solutions. The last column gives the percentage of Pareto solutions that improves the ECS solution on the two objectives simultaneously. As indicated in the last line of the table, the hybrid approach provided **19.77** solutions on average, and **83.04%** of the Pareto solutions found by this hybrid approach improve the ECS solution on the two objectives simultaneously. In addition, Table 6 shows that the more tasks there are, the more Pareto solutions are found, and the more the percentage of Pareto solutions dominating the ECS solution increases.

Table 6: Comparison of Pareto hybrid approach solutions and ECS solution

		Average number of Pareto solutions	Pareto solutions dominating ECS solution (%)
Number of tasks	∼20	14.78	78.24
	∼40	19.57	80.70
	∼60	21.36	83.62
	∼80	21.45	83.12
	∼120	21.67	89.51
Number of processors	02	18.51	73.21
	04	19.42	71.01
	08	22.17	75.83
	16	23.32	86.12
	32	19.98	94.73
	64	15.18	97.36
Heterogeneity	100	20.12	80.82
	200	21.67	74.47
	random	17.52	93.83
CCR	0.1	19.60	88.44
	0.2	19.47	88.83
	1.0	17.53	89.09
	5.0	19.26	78.80
Average		**19.77**	**83.04**

To determine the contribution of the new approach, in terms of the values of makespan and energy consumption, we compare the solution provided by ECS to only one solution of the Pareto set provided by the new approach. The solution chosen in the Pareto set is used only to compare the new approach with ECS. Nevertheless the decision maker, using the new approach, will have a set of Pareto solutions instead of one solution.

For each instance,

- a first resolution is done using ECS to provide one solution s.
- a second resolution is done using the new approach to obtain a set E of Pareto solutions.
- only one Pareto solution s' is selected from the set E. This solution is the closest to s in the sense of Euclidean distance.
- finally, a comparison will be done between the solutions s and s'.

Figure 10, Figure 11, Figure 12 and Table 7 allow to compare in a detailed way the two approaches. They respectively show the improvement brought by the new approach according to the number of tasks, the number of processors, the CCR, and the processor heterogeneity. Experiments show that our approach improves on average the results obtained by ECS. Indeed, as shown in Table 7, the energy consumption is reduced by **47.49%** and the makespan reduced by of **12.05%**. In addition, Figure 11 shows clearly that the more processors there are, the more the new approach improves the results of ECS.

Table 7: Improvement according to the processor heterogeneity

Processor heterogeneity	Energy (%)	Makespan (%)
100	44.74	9.10
200	43.49	7.07
random	54.26	19.99
Average	**47.49**	**12.05**

5.3. Insular approach

The objective of the following experiments is to show that our island approach improves the quality of the solutions provided by the hybrid approach. This insular approach is useful when solving large instances. Therefore, the

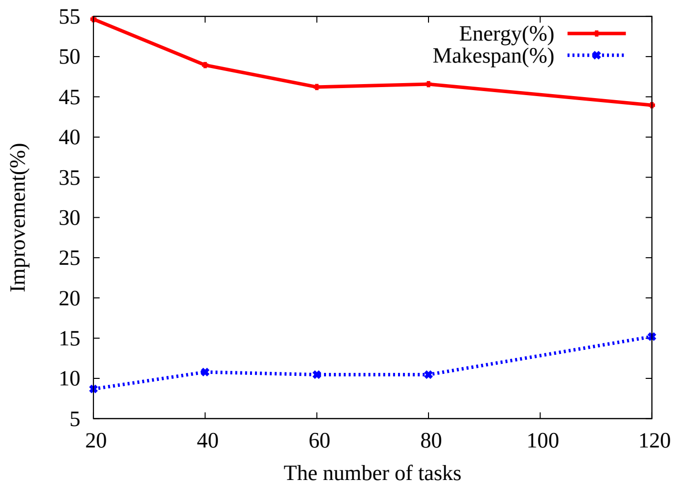

Figure 10: Improvement according to the number of tasks

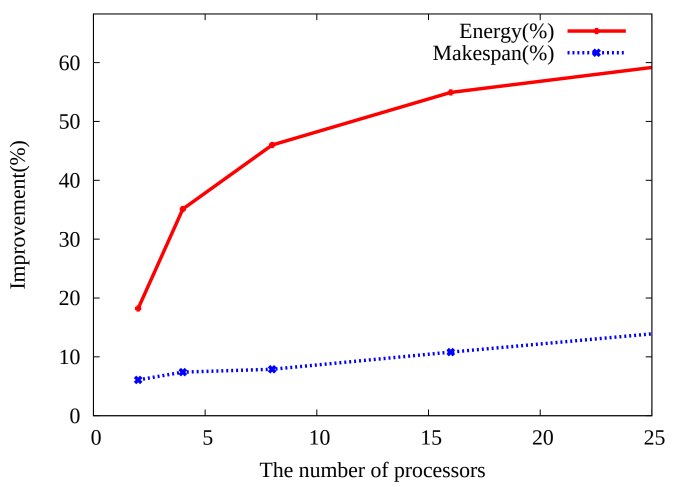

Figure 11: Improvement according to the number of processors

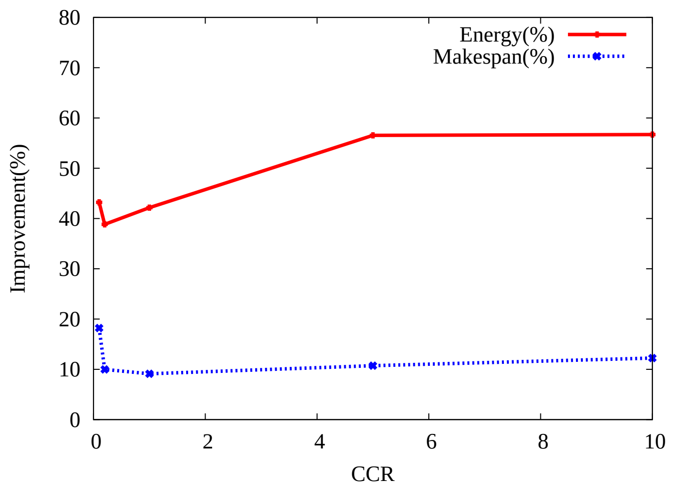

Figure 12: Improvement according to the CCR

experiments, presented in this section, focus only on the large instances of Table 4. The instances used are those with the number of tasks is 120, the number of processors is 64, the value of CCR is 10, and the heterogeneity of processors is 200 (20 instances). Each instance is solved using 1, 5, 10, 30 or 50 islands. An insular approach with 1 island is equivalent to the hybrid approach.

Figure 13 illustrates the S-metric average values obtained with different numbers of islands. These values are normalized with the average value obtained by the experiments using 1 island. The S-metric measures the hyper-volume defined by a reference point and a Pareto front. It allows to evaluate the quality of a Pareto front provided by an algorithm.

Experiments show that whatever the number of used islands the insular approach improves the Pareto front obtained with the hybrid approach. As shown in Figure 13, the use of 50 islands, instead of 1 island (i.e. the hybrid approach), improves the S-metric of the obtained Pareto front by **26%**. In Figure 13, the more the number of islands is used, the better the results will be.

5.4. Multi-start approach

This section presents the experiments done to assess the quality of our multi-start approach. The parameters of the instances used in our experiments are: the CCR is 0.1, 0.5, 1.0, 5.0 or 10.0, the number of processors is

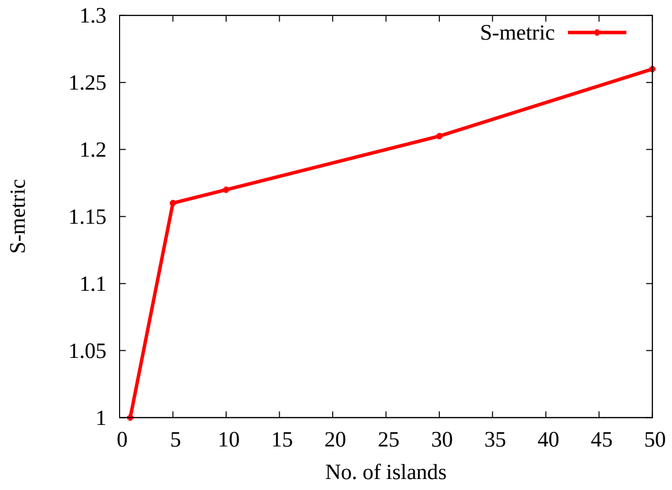

Figure 13: S-metric value according to the number of islands

8, 32, or 64, and the heterogeneity of processors is 100, 200 or random. The population of the GA contains 20 chromosomes. Therefore, 21 computing cores are used to solve each instance (20 cores to run the ECSs and 1 core to run the GA). In our case, an experiment can not have a speedup greater than 21.

Table 8, Figure 14, Figure 15 show respectively the evolution of the speedup according to the processor heterogeneity, the CCR, and the number of processors. The average speedup obtained is 13.06.

As shown in Figure 15, the speedup increases proportionally to the number of processors on which the precedence-constrained parallel application is run. Table 8 and Figure 15 that show the CCR and the heterogeneity of processors do not impact significantly the quality of the acceleration of our approach.

6. Conclusions

In this paper, we have investigated the precedence-constrained parallel applications particularly on high-performance computing systems like cloud computing infrastructures. Precedence-constrained parallel applications are designed mostly with the sole goal of minimizing completion time without paying much attention to energy consumption.

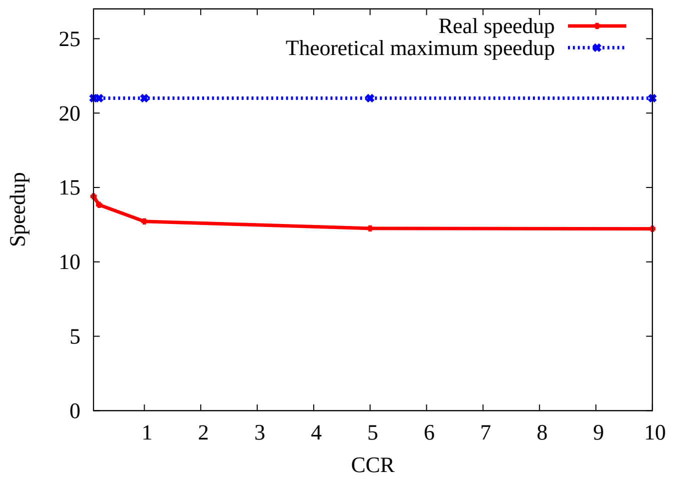

Figure 14: Speedup according to the CCR

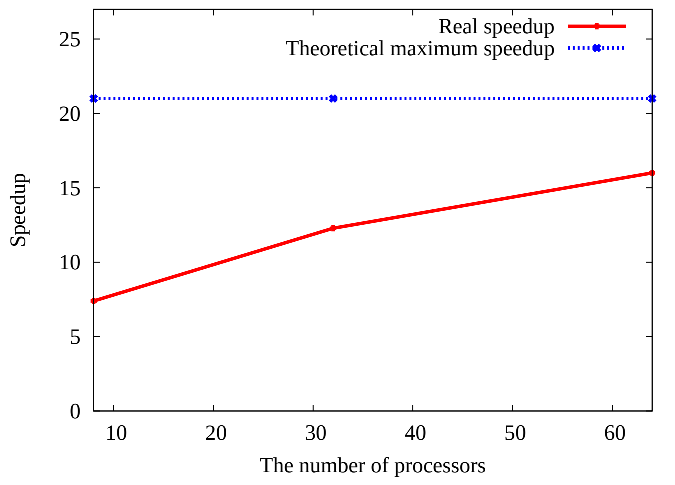

Figure 15: Speedup according to the number of processors

Table 8: Speedup according to the processor heterogeneity

Processor heterogeneity	Speedup
100	13.74
200	12.73
random	12.73
Average speedup	**13.06**

We presented a new parallel bi-objective hybrid genetic algorithm to solve this problem. The algorithm minimizes energy consumption and makespan. The energy saving of our approach exploits the dynamic voltage scaling (DVS) technique—a recent advance in processor design.

Our new approach has been evaluated with the Fast Fourier Transformation task graph which is a real-world application. Experiments show that our bi-objective meta-heuristic improves on average the results obtained in the literature (see [25], [1] and [9]) particularly in energy saving. Indeed, the energy consumption is reduced by **47.5%** and the completion time by **12%**. The experiments of the insular approach also show that the more the number of islands is used, the better the results will be. The use of 50 islands, instead of 1 island (i.e. the hybrid approach), improves the S-metric of the obtained Pareto front by **26%**. Furthermore, the multi-start approach is on average **13** times faster than the island approach using 21 cores.

However, we observed that the hybrid approach consumes more resources than ECS, and the insular approach consumes more resources than the hybrid approach. In the insular approach, experiments show that the more the number of islands is used, the more the resources are needed. A resource can be a processor, a network bandwidth, etc. The energy consumed by an approach increases when the used resources increase. We think that the multi-start approach does not increase significantly the energy consumed by the insular approach.

Therefore, one of the main perspectives of the work presented in this paper is to determine the solving approach to choose among ECS, the hybrid approach, and the insular approach, according to the precedence-constrained parallel application at hand. If the insular approach is chosen, the major issue is to determine the best number of islands to be used. This future work aims to minimize the total amount of consumed energy by the chosen solving approach and by the precedence-constrained parallel application to be solved. It is clear, for example, that the insular approach is interesting for the large and resource consuming precedence-constrained parallel applications and the applications intended to be executed several times.

Appendix

Algorithm 1 The main parameters of our approaches

```
 1: INSULAR←TRUE
 2: MULTISTART←TRUE
 3:
 4: GENERATION_MAXIMUM←200
 5: POPULATION_SIZE←20
 6: CROSSOVER_RATE←1
 7: MUTATION_RATE←0.35
 8: N←size(solution)
 9:
10: MIGRATION_TOPOLOGY←RING
11: MIGRATION_RATE←20
12: MIGRANTS_SIZE←5
```

Algorithm 2 The main algorithm of our approaches

```
hybrid_insular_multistart_approches()
 1: initialize(population,POPULATION_SIZE)
 2: evaluate(population)
 3: GENERATION←1
 4: while GENERATION ≤ GENERATION_MAXIMUM do
 5:    parents←select(population)
 6:    children←crossover(parents)
 7:    mutation(children)
 8:    evaluate(children)
 9:    replace(population,children)
10:    update(archive,children)
11:    migrate(population,GENERATION)
12:    GENERATION←GENERATION+1
13: end while
```

Algorithm 3 Fitness operator

```
evaluate(population)
1: if MULTISTART then
2:     evaluate_parallel(population)
3: else
4:     evaluate_sequential(population)
5: end if
evaluate_parallel(population)
1: for all solution $\in$ population do
2:     launch_parallel(ECS_component2,solution)
3: end for
4: for all solution $\in$ population do
5:     cost(solution)$\leftarrow$read_cost(solution)
6: end for
evaluate_sequential(population)
1: for all solution $\in$ population do
2:     cost(solution)$\leftarrow$ECS_component2(solution)
3: end for
```

Algorithm 4 Mutation operator

```
mutation(children)
1: for all solution $\in$ children do
2:     if random([0,1]) $\leq$ MUTATION_RATE then
3:         mutation(solution)
4:     end if
5: end for
mutation(solution)
1: (i,j)=random(1$\leq$i<j$\leq$N $\wedge$ check_level(solution,i,j))
2: swap(task(solution$_i$),task(solution$_j$))
check_level(solution,i,j)
1: return b-level(task(solution$_i$))<b-level(task(solution$_j$))
```

Algorithm 5 Crossover operator

```
crossover(parents)
 1: children ← $\emptyset$
 2: for all i $\in$ (1...POPULATION_SIZE) do
 3:    if random([0,1]) $\leq$ CROSSOVER_RATE then
 4:       (parent1,parent2)←select(parents)
 5:       (child1,child2)←crossover2(parent1,parent2)
 6:       add(children,child1)
 7:       add(children,child2)
 8:    end if
 9: end for
10: return children
crossover2(parent1,parent2)
 1: child1←crossover1(parent1,parent2)
 2: child2←crossover1(parent2,parent1)
 3: return (child1,child2)
crossover1(parent1,parent2)
 1: (i,j)=random(1$\leq$i<j$\leq$N)
 2: for all k $\in$ (1,...,i-1,j+1,...,N) do
 3:    task(child$_k$)←task(parent1$_k$)
 4: end for
 5: m←0
 6: for all [k $\in$ (1,..,N)] $\wedge$ [task(parent2$_k$) $\notin$ tasks(child)] do
 7:    task(solution_buffer$_m$)←task(parent2$_k$)
 8:    m←m+1
 9: end for
10: for all k $\in$ (i,...,j) do
11:    task(child$_k$)←task(solution_buffer$_{k-i+1}$)
12: end for
13: return child
```

Algorithm 6 Migration operator

```
migrate(population,GENERATION)
1: if NOT INSULAR then
2:    stop
3: end if
4: if GENERATION mod(MIGRATION_RATE)≠0 then
5:    stop
6: end if
7: migrants←select(population,MIGRANTS_SIZE)
8: DESTINATION←destination(topology)
9: send(DESTINATION,migrants)
10: SOURCE←source(topology)
11: migrants←receive(SOURCE)
12: insert(population,migrants)
```

Acknowledgments

Professor Albert Zomaya's work is support by an Australian Research Grant DP1097110. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). Clusters of University of Mons were also used. We would like to thank the technical staffs of the Grid'5000 and the clusters of University of Mons for making their clusters accessible and fully operational.

References

[1] D. Bozdag, U. Catalyurek, and F. Ozguner. A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, 2006.

[2] D.P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12(5):489–500, 2009.

[3] K.W. Cameron. Trading in green it. *Computer*, 43(3):83–85, March 2010.

[4] J.J. Chen and T.W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. pages 13–20, June 2005.

[5] J.P. Cohoon, S.U. Hegde, W.N. Martin, and D. Richards. *Punctuated equilibria: A parallel genetic algorithm.* 1987.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms.* The MIT press, 2001.

[7] S. Darbha and D.P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 9(1):87–95, 2002.

[8] K. Deb. *Multi-objective optimization using evolutionary algorithms.* Wiley, 2001.

[9] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* WH Freeman and Company, 1979.

[10] R. Ge, X. Feng, and K.W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 34–44, November 2005.

[11] Intel. Intel pentium m processor datasheet, 2004.

[12] S.U. Khan and I. Ahmad. A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. *IEEE Transactions on Parallel and Distributed Systems*, 20(3):346 –360, March 2009.

[13] K.H. Kim, R. Buyya, and J. Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In *Proceedings of the* 7^{th} *IEEE International Symposium on Cluster Computing and the Grid*, pages 541–548, May 2007.

[14] S.C. Kim, S. Lee, and J. Hahm. Push-pull: Deterministic search-based dag scheduling for heterogeneous cluster systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(11):1489–1502, 2007.

[15] G. Koch. Discovering multi-core: Extending the benefits of Moore's law. *Technology Intel Magazine*, page 1, July 2005.

[16] J.G. Koomey. Estimating total power consumption by servers in the US and the world, 2007.

[17] Y.K. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. In *Proceedings of the IEEE* 1^{st} *Merged International Parallel Symposium/Symposium on Parallel and Distributed Processing (IPPS/SPDP)*, pages 531–537, 1998.

[18] Y.C. Lee and A.Y. Zomaya. A productive duplication-based scheduling algorithm for heterogeneous computing systems. *Proceedings of International Conference on High Performance Computing and Communications (HPCC)*, pages 203–212, 2005.

[19] Y.C. Lee and A.Y. Zomaya. A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 19(9):1215 – 1223, September 2008.

[20] Y.C. Lee and A.Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Proceedings of the* 9^{th} *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pages 92–99, 2009.

[21] A. Liefooghe, L. Jourdan, and Talbi E-G. A software framework based on a conceptual unified approach for evolutionary multiobjective optimization: ParadisEO-MOEO. *European Journal of Operational Research (EJOR)*, 209(2):104–112, 2011.

[22] R. Min, T. Furrer, and A. Chandrakasan. Dynamic voltage scaling techniques for distributed microsensor networks. In *Proceedings of IEEE Workshop on VLSI*, pages 43–46, April 2000.

[23] B. Rountree, D.K. Lowenthal, S. Funk, V.W. Freeh, B.R. de Supinski, and M. Schulz. Bounding energy consumption in large-scale MPI programs. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 1–9, November 2007.

[24] E.G. Talbi. *Metaheuristics: From design to implementation.* Wiley, 2009.

[25] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.

[26] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys (CSUR)*, 37(3):195–237, September 2005.

[27] L. Wang, G. von Laszewski, J. Dayal, and F. Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In *Proceedings of The* 10^{th} *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'10)*, pages 368–377, May 2010.

[28] X. Zhong and C.Z. Xu. Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. *IEEE Transactions on Computers*, 56(3):358–372, 2007.

[29] D. Zhu, R. Melhem, and B.R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(7):686–700, 2003.

[30] D. Zhu, D. Mosse, and R. Melhem. Power-aware scheduling for and/or graphs in real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):849–864, September 2004.

[31] A.Y. Zomaya, C. Ward, and B. Macey. Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems*, 10(8):795–812, August 1999.