Received <day> <Month>, <year>; | Revised <day> <Month>, <year>; | Accepted <day> <Month>, <year>

DOI: xxx/xxxx

ARTICLE TYPE

Energy-aware Load Balancing of Parallel Evolutionary Algorithms with Heavy Fitness Functions in Heterogeneous CPU-GPU Architectures

Juan José Escobar* | Julio Ortega | Antonio Francisco Díaz | Jesús González | Miguel Damas

This is the peer reviewed version of the following article: *Energy-aware Load Balancing of ParallelEvolutionary Algorithms with Heavy Fitness Functions in Heterogeneous CPU-GPU Architectures*. In: Concurrency and Computation: Practice and Experience 31.6 (2019), e4688, which has been published in final form at https://doi.org/10.1002/cpe.4688. This article may be used for non-commercial purposes in accordance with Wiley Terms and Conditions for Use of Self-Archived Versions

Department of Computer Architecture and Technology, CITIC, University of Granada, Spain

Correspondence
*Juan José Escobar, Email: jjescobar@ugr.es

Present Address
Calle Periodista Rafael Gómez Montero, 2, 18014, Granada, Spain

Abstract

By means of the availability of mechanisms such as Dynamic Voltage and Frequency Scaling (DVFS) and heterogeneous architectures including processors with different power consumption profiles, it is possible to devise scheduling algorithms aware of both runtime and energy consumption in parallel programs. In this paper, we propose and evaluate a multi-objective (more specifically, a bi-objective) approach to distribute the workload among the processing cores in a given heterogeneous parallel CPU-GPU architecture. The aim of this distribution may be either to save energy without increasing the running time or to reach a trade-off among time and energy consumption. The parallel programs considered here are master-worker evolutionary algorithms where the evaluation of the fitness function for the individuals in the population demands the most part of the computing time. As many useful bioinformatics and data mining applications exhibit this kind of parallel profile, the proposed energy-aware approach for workload scheduling could be frequently applied.

KEYWORDS:
Dynamic Voltage and Frequency Scaling (DVFS); Energy-aware Scheduling; Heterogeneous CPU-GPU Parallel Architectures; Load Balancing; Master-worker Algorithms; Modeling of Energy Consumption

1 | INTRODUCTION

Nowadays, due to economic and environmental reasons, energy-saving has emerged as one of the main goals in computing systems and the term efficiency not only means good speedups but also optimal energy consumption. Thus, the development of scheduling procedures that take into account both objectives constitutes an important research issue (1, 2, 3, 4).

In this regard, two main approaches could be explored. On the one side, present microprocessors usually implement power management policies that usually change between microprocessors and are not visible to the user. In this case, it is necessary to devise a black-box approach (5) that models the main characteristics of the applied power management policy. The other approach uses techniques, such as DVFS, that enable the user to control the frequency and voltage of the processors in the platform (2, 3). In both cases, the target is the development of a task scheduling procedure that, by being aware of those factors, minimizes runtime and reduces energy consumption.

This multi-objective requirement for task scheduling procedures is tackled in the present paper by proposing a multi-objective (i.e. a bi-objective) cost function that represents the user choice instead of using a Pareto-based multi-objective approach. This cost function substitutes the user's selection of one of a non-dominated solution in the front obtained from a Pareto-based

[0]**Emails:** jjescobar@ugr.es, jortega@ugr.es, afdiaz@ugr.es, jesusgonzalez@ugr.es, mdamas@ugr.es

multi-objective approach to the optimization problem by the assignment of a suitable value to the parameters that define the proposed bi-objective cost function. To build the cost function, suitable models are needed to predict both running time and energy consumption corresponding to the workload to be distributed. One of the contributions of this paper is an approach based on linear regression to build the energy model from experimental measures obtained from the energy consumption of the computing node in the development of parallel master-worker evolutionary algorithms implemented in heterogeneous CPU-GPU architectures. Our experiments in this paper also show that, for many applications, the effect of programming strategies that take into account energy consumption principles is directly observable in the consumption of the computing nodes.

After this introduction, Section 2 describes the characteristics and applications of parallel evolutionary algorithms considered in the paper. Then, Section 3 describes the proposed cost function including the goals of minimizing runtime and energy consumption. It can be used to implement scheduling procedures that provide the allocation of tasks to processors to reach different trade-offs between runtime and energy consumption according to the relative weight assigned to each one, as it is illustrated in Section 4 in case of CPU-GPU platforms and master-worker parallel evolutionary algorithms. The experimental results to evaluate our proposal are provided in Section 5, and Section 6 summarizes the contributions published on energy-aware scheduling procedures described in the literature and related with the work here presented. Finally, the conclusions are given in Section 7.

2 | MASTER-WORKER PARALLEL EVOLUTIONARY ALGORITHMS

Many bioinformatics and data mining applications comprise tasks, such as classification, clustering, feature selection, and optimization, that implement metaheuristics based on evolutionary algorithms. As these algorithms are inspired in the natural evolution, they evolve a set of solutions (population of individuals) for the target problem across a given number of iterations (generations) by applying operators, such as selection, mutation and/or crossover to the solutions in the population according to their quality, or fitness. This way, for a given iteration or generation, an evolutionary algorithm has to evaluate the fitness of the solutions in the population according to a cost, or performance function. Once the solutions are evaluated, a selection operator is applied to include the best solutions in the population for the next generation, and new solutions are also generated to complete this new population by changing some of these solutions (through a mutation operator), by combining several solutions (through a crossover operator) (6). Thus, each generation of an evolutionary algorithm requires the fitness evaluation of the solutions in the population. This could imply a significant runtime, in many real problems where the fitness function is costly to compute and/or in algorithms that evolve a population with a large number of solutions. Therefore, parallel processing constitutes a useful approach to accelerate the execution of evolutionary algorithms and/or to improve the quality of the solutions they provide as greater populations can be evolved in the same amount of times as more processors are involved. Different approaches can be considered to parallelize an evolutionary algorithm, as can be seen in (7). In this way, it is possible to implement in parallel the fitness evaluation for the different solutions in the population, or to distribute the solutions among smaller subpopulations that evolve independently and interchange solutions after a given number of generations. Moreover, different profiles of parallelism are present in the corresponding codes that evaluate the fitness for a given solution and it is interesting to develop efficient parallel codes for the present heterogeneous CPU-GPU platforms.

With respect to bioinformatics and data mining applications, for example in (8), an evolutionary multi-objective optimization is applied to solve a feature selection problem in a Brain Computer Interface (BCI) application. The individuals of the population correspond to different sets of features that define the components of the patterns to be classified. These sets of features have to be evaluated by the accuracy and generalization capabilities of the classifier once it has been adjusted by using the training patterns characterized by the selected features. The iterations required to train the classifier usually require a high amount of computing time as it is shown in Table 1 , which provides the runtime for the steps of a feature selection procedure based on an evolutionary algorithm, described in (9) and analyzed with the tool *gprof* (10). The fitness evaluation needs between 99.93% (with 120 individuals in the population) and 98.60% of the runtime (with 15,000 individuals). Moreover, as the evaluation of the fitness is completely independent for each individual in the population, a master-worker approach constitutes a very suitable alternative to efficiently parallelize this kind of problems. This way, according to Amdahl's law, our hypothesis in this paper considers that an energy-aware procedure to allocate the fitness evaluation tasks is still useful to devise an energy-efficient master-worker evolutionary algorithm because, even in the case of a very large number of individuals in the population, the percentage of runtime devoted to evaluate the fitness of the population represents the most time consumed of the procedure (above 98%).

Thus, this paper tackles energy consumption of parallel programs whose tasks dependence graphs are shown in Figure 1 (a). In this graph, tasks $T1, ..., TN$ can be executed in parallel after task $T0$, and after synchronizing themselves once they have finished,

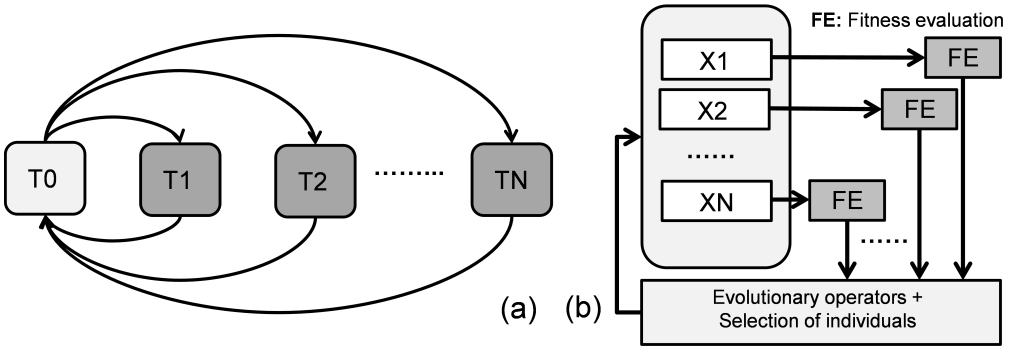


FIGURE 1 Task dependence graph considered (a), and evolutionary algorithm as example of application with such graph (b)

task $T0$ is executed again and generates another set of parallel tasks, $T1, ..., TN$, executed in parallel, and so on. Moreover, the runtime of task $T0$ is negligible with respect to runtime of each parallel task, $T1, ..., TN$. Many useful applications can be parallelized according to the dependence graph of Figure 1 (a). Indeed, Figure 1 (b) schematizes an evolutionary algorithm that whose main characteristics have been previously summarized. In each generation, the fitness of the individuals in the population has to be evaluated according to some performance procedure that could demand a costly computation.

3 | A COST FUNCTION FOR MULTI-OBJECTIVE WORKLOAD SCHEDULING

The scheduling strategy proposed in the paper have to take into account information about both goals of runtime and energy consumption reduction. These two objectives usually correspond to opposed goals, as improving runtime could imply increasing the instantaneous power. An increment in the instantaneous power across the program runtime could correspond either to an increment or a decrement in the energy consumed by the program, depending on the achieved acceleration on the runtime. This way, we have to cope with a multi-objective (in this case a bi-objective) problem (11). An alternative to solve it is to use an optimization algorithm that searches a set of Pareto non-dominated solutions (non-dominated scheduling alternatives) among which the user selects the most appropriate one for the given situation. Nevertheless, due to the runtime required by this Pareto-based multi-objective approach, in this paper we use a cost function whose minimum corresponds to the desired trade-off among the two considered goals of time and energy. Of course, this trade-off is set offline by setting the values of the corresponding parameters of the bi-objective cost function).

Therefore, given a task i with a workload equal to C_i clock cycles, that have been allocated to a processor j running at frequency f_j, it is possible to define two indices, $\Delta t(C_i, f_j)$ and $\Delta E(C_i, f_j)$, with values between 0 and 1. These indices are respectively related with the relative deviations of the estimated time and energy consumption, $t(C_i, f_j)$ and $E(C_i, f_j)$, for the allocation of a task C_i to the processor j running at frequency f_j, as is given in Equation (1):

$$\begin{aligned} \Delta t(C_i, f_j) &= \frac{t(C_i, f_j) - t(C_{min}, f_{max})}{t(C_{max}, f_{min}) - t(C_{min}, f_{max})} \\ \Delta E(C_i, f_j) &= \frac{E(C_i, f_j) - E(C_{min}, f_{min})}{E(C_{max}, f_{max}) - E(C_{min}, f_{min})} \end{aligned} \tag{1}$$

This way, to select a processor and the corresponding frequency for a given task, the scheduling algorithm could use a cost function that takes into account both the energy and runtime objectives through indices $\Delta t(C_i, f_j)$ and $\Delta E(C_i, f_j)$. Here, we propose a bi-objective cost function $\Delta(C_i, f_j) = a \cdot \Delta t(C_i, f_j) + b \cdot \Delta E(C_i, f_j)$ with positive coefficients a and b verifying that $a + b = 1$. This cost function promotes allocations with low values of $\Delta t(C_i, f_j)$ and $\Delta E(C_i, f_j)$, and even lower values for one factor whenever the other factor grows. Depending on the relative values of a and b, it is possible to give more relevance either to optimize runtime or energy consumption.

To obtain the values of the deviations $\Delta t(C_i, f_j)$ and $\Delta E(C_i, f_j)$ for a given task i, on which the multi-objective cost function depends, the corresponding models, $t(C_i, f_j)$ and $E(C_i, f_j)$, that relate the runtime and the energy consumption with the workload C_i and frequencies f_j, are required. These models also allow the determination of the maxima and minima values for runtime and energy consumption: $t(C_{max}, f_{min})$, $E(C_{max}, f_{max})$, $t(C_{min}, f_{max})$, and $E(C_{min}, f_{min})$.

Different approaches have been considered to estimate the models $t(C_i, f_j)$ and $E(C_i, f_j)$ that quantify the time and energy consumption behaviours of the program execution in the given computing platform. For example, in (12), the target application is run for different configurations in order to be able to fit a regression model to the observations. Other example of this black-box approach is shown in (5). As in (3) and many other previous work dealing with this issue, the energy model can be also estimated from the power consumption equations corresponding to CMOS circuits that include the terms associated to capacitive, short-circuit, and leakage power. This way, assuming the capacitive term as the most significant one, we could estimate the power consumption in a processor as:

$$Pow_j = \beta \cdot f_j \cdot {V_j}^2 \tag{2}$$

Where parameter β is related with the product of the number of transistors switching in the processor per clock cycle and the total capacitance load, f_j is the clock frequency, and V_j is the supply voltage of the processor j. Therefore, the energy E_j consumed by a given task i that requires C_i clock cycles in a processor with a supply voltage V_j can be estimated from Equation (2) by:

$$\begin{aligned} E_j &= \beta \cdot f_j \cdot {V_j}^2 \cdot \frac{C_i}{f_j} \\ &= \beta \cdot {V_j}^2 \cdot C_i \end{aligned} \tag{3}$$

Whenever a processor is idle, there is also a so-called indirect energy consumption term that for a given processor j can be estimated by:

$$E_j^{idle} = \beta \cdot f_j \cdot \left(V_j^{idle}\right)^2 \cdot t_j \tag{4}$$

being V_j^{idle} the supply voltage of the processor in its idle state and t_j is the amount of time in which processor j has been in this state. The tasks have to be allocated to the processors included in a heterogeneous platform with N_P processors, $P_j (j = 1, ..., N_P)$. Each processor j can operate at different Voltage Supply Levels (VSLs), $V_{j,l} (l = 1, ..., \omega(j))$, corresponding to different clock frequencies $f_{j,l} (j = 1, ..., N_P)(l = 1, ..., \omega(j))$. This way, if we use these models to estimate the energy consumption, and considering $\frac{C_i}{f_{j,l}}$ as the runtime of a task i with workload C_i allocated to processor j running at frequency $f_{j,l}$, the relative deviations of Equation (1) can be given as:

$$\Delta t(C_i, f_{j,l}) = \frac{\frac{C_i}{f_{j,l}} - t(C_{min}, f_{max})}{t(C_{max}, f_{min}) - t(C_{min}, f_{max})} \tag{5}$$

$$\Delta E(C_i, f_{j,l}) = \frac{\beta \cdot {V_{j,l}}^2 \cdot C_i - E(C_{min}, f_{min})}{E(C_{max}, f_{max}) - E(C_{min}, f_{min})} \tag{6}$$

In Equation (5), the parameters, $t(C_{max}, f_{min})$ and $t(C_{min}, f_{max})$ can be obtained from the highest and lowest clock cycle values, C_i, required to complete the estimated workloads of the different tasks $T_i (i = 1, ..., N)$ and from the frequencies of the available processors, $f_{j,l}$. In Equation (6), the energy consumed while the processors are idle is not explicitly shown to prevent unnecessary complexities in the mathematical expressions. Thus, the following parameters can be evaluated:

$$\begin{aligned} t(C_{max}, f_{min}) &= \frac{max\left(C_i\right)}{min\left(f_{j,l}\right)} \\ t(C_{max}, f_{max}) &= \frac{max\left(C_i\right)}{max\left(f_{j,l}\right)} \\ t(C_{min}, f_{min}) &= \frac{min\left(C_i\right)}{min\left(f_{j,l}\right)} \\ t(C_{min}, f_{max}) &= \frac{min\left(C_i\right)}{max\left(f_{j,l}\right)} \end{aligned} \tag{7}$$

Parameters $t(C_{max}, f_{min})$, $t(C_{max}, f_{max})$, $t(C_{min}, f_{min})$, and $t(C_{min}, f_{max})$ verify that $t(C_{min}, f_{max}) < t(C_{max}, f_{max}) < t(C_{max}, f_{min})$ and $t(C_{min}, f_{max}) < t(C_{min}, f_{min}) < t(C_{max}, f_{min})$. The parameter $t(C_{max}, f_{min})$ is the time required by the task with the heaviest workload when it is executed in a processor running at the lowest frequency. The parameter $t(C_{max}, f_{max})$ is the time required by the task with the highest workload in a processor running at the highest frequency. This way, $t(C_{max}, f_{min})$ and $t(C_{max}, f_{max})$ respectively represent the highest and lowest running times that the heaviest task would require in the present heterogeneous

platform. In the same way, $t(C_{min}, f_{min})$ and $t(C_{min}, f_{max})$ are, respectively, the highest and lowest running times for the lightest task. In Equation (5) only $t(C_{max}, f_{min})$ and $t(C_{min}, f_{max})$ are required. It is also possible to define energy consumption parameters, $E(C_{max}, f_{min})$, $E(C_{max}, f_{max})$, $E(C_{min}, f_{min})$, and $E(C_{min}, f_{max})$, that respectively correspond to the runtimes $t(C_{max}, f_{min})$, $t(C_{max}, f_{max})$, $t(C_{min}, f_{min})$, and $t(C_{min}, f_{max})$ as follows:

$$\begin{aligned} E(C_{max}, f_{min}) &= max\left(C_i\right) \cdot \beta \cdot min\left(V_{j,l}\right)^2 \\ E(C_{max}, f_{max}) &= max\left(C_i\right) \cdot \beta \cdot max\left(V_{j,l}\right)^2 \\ E(C_{min}, f_{min}) &= min\left(C_i\right) \cdot \beta \cdot min\left(V_{j,l}\right)^2 \\ E(C_{min}, f_{max}) &= min\left(C_i\right) \cdot \beta \cdot max\left(V_{j,l}\right)^2 \end{aligned} \tag{8}$$

This way, $E(C_{max}, f_{min})$, $E(C_{max}, f_{max})$, $E(C_{min}, f_{min})$, and $E(C_{min}, f_{max})$ verify that $E(C_{min}, f_{min}) < E(C_{max}, f_{min}) < E(C_{max}, f_{max})$ and $E(C_{min}, f_{min}) < E(C_{min}, f_{max}) < E(C_{max}, f_{max})$. In Equation (6), for the relative deviation in the energy consumption, only $E(C_{max}, f_{max})$ and $E(C_{min}, f_{min})$ are required.

In what follows, we describe a scheduling procedure that allocates processors and frequencies to tasks trying to minimize both runtime and energy consumption by taking advantage of the proposed bi-objective cost function that joins the deviations of Equation (1). Such scheduling procedure should allocate tasks on the available processors according to the available models about their computational cost and energy consumption. The procedure also needs information about the characteristics of the processors in the system where the tasks will be executed. Among these characteristics, the possible frequencies at which the processors can run, and the availability of changing frequencies and voltages either by the operating system or the user, determine the applicability of static or dynamic scheduling procedures and their implementation inside the application or in the runtime system. Algorithm 1 provides the proposed scheduling procedure. Given a set of tasks with workloads C_i, the procedure sorts the tasks according to their workloads, verifying that $C_i \leq C_{i+1}$. The N_P processors among which the tasks have to be distributed are also sorted according to the values of their frequencies so that, for processor j-th, its FL possible operating frequencies verify that $f_{j,1} \leq f_{j+1,1} (j = 1, ..., N_P - 1)$ and $f_{j,k} \geq f_{j,k+1} (k = 1, ..., FL - 1)$.

The procedure of Algorithm 1 proceeds as follows. Given the first task, C_1, in the sorted list of tasks, the cost function is evaluated for the allocation of this task to all possible frequencies in the available processors, $\Delta(C_1, f_{j,k}) = a \cdot \Delta t(C_1, f_{j,k}) + b \cdot \Delta E(C_1, f_{j,k})$. The task is allocated to the processor/frequency for which the lowest value of $\Delta(C_1, f_{j,k})$ is obtained. Then, the processor (and its operating frequencies) is removed from the list of available processors and the procedure continues with the next task in the sorted list. Given a set of N_T tasks to be distributed among N_P processors, each with FL possible frequencies, the complexity of the procedure is $\theta({N_P}^2 \cdot FL)$. The procedure implements a greedy algorithm that only ensures to get local optima, although it would be possible to use more powerful optimization procedures such as an evolutionary algorithm, this would requires an unsuitable higher computational cost.

The energy-aware scheduling procedure of Algorithm 1 could be implemented either in the runtime system or in the application, depending on whether the changes in the processor's frequency are done at system or user level. Thus, it could be implemented inside the application code whenever DVFS is available at user level. Moreover, the procedure can be also implemented in heterogeneous platforms including processors with different energy consumption profiles (i.e. different operating frequencies or voltages) although it would not be able to change frequencies or voltages. This situation corresponds to having processors operating at different constant frequencies: $f_{1,1}, ..., f_{p,1}$ with $f_{j,1} \neq f_{k,1}$ for at least two different processors j and k.

Figure 2 provides information about the increments in runtime and decrements in energy consumption obtained by simulating the procedure described in Algorithm 1 for 100 different randomly selected configurations of tasks with computing costs between 100 and 3,000 cycles and cost differences multiple of 100 cycles. A heterogeneous configuration with processors corresponding to two different sets of relative speeds has been considered as is shown in Table 2 , which provides the relative speeds with respect to the one achieved at the highest frequency of 1 GHz (100% in P5, P6, P7 and P8). The second frequency in Table 2 for each processor is an eighty percent of its highest frequency and the lowest frequency is one half of this highest frequency. The highest frequency for the first type of processors is an eighty per cent of the highest frequency for the other group of processors. An idle frequency of 100 MHz has been considered in the simulations and the (a, b) parameter couples simulated are $(0.1, 0.9)$, $(0.25, 0.75)$, $(0.5, 0.5)$, $(0.75, 0.25)$, and $(0.9, 0.1)$. Figure 2 (a) compares the scheduling obtained by the procedure of Algorithm 1 with a random allocation of tasks to the processors running at their highest frequencies. Figure 2 (b) shows the increments with respect to a scheduling that provides the minima runtimes. Figure 2 demonstrates that, as the coefficient b increases (and as a consequently decreases), the decrements (with respect to the scheduling used as reference) in energy consumption are larger at the cost of increasing the runtimes.

Algorithm 1 Description of the energy-aware procedure for scheduling

Function `Energy_Aware_Scheduling` $\left(C, N_P, f, FL, a, b\right)$

Input : Set C: Cycles of task i to be allocated to a processor, $C_i; \forall i = 1, ..., N_P$
Input : Number of processors, N_P
Input : Set f: Frequency i^{th} of the processor j^{th}, $f_{i,j}; \forall i = 1, ..., FL; j = 1, ..., N_P$
Input : Number of frequency levels, FL
Input : First coefficient of the cost function, a, related with the execution time
Input : Second coefficient of the cost function, b, related with the energy consumption
Output: S, the new solution for the problem

1 C_{max} and $C_{min} \leftarrow$ `max` $\left(C_i\right)$ and `min` $\left(C_i\right)$, respectively
2 f_{max} and $f_{min} \leftarrow$ `max` $\left(f_{i,j}\right)$ and `min` $\left(f_{i,j}\right)$, respectively
3 $t(C_{max}, f_{min})$ and $t(C_{min}, f_{max}) \leftarrow$ Compute Equation (7)
4 $E(C_{max}, f_{min})$ and $E(C_{min}, f_{max}) \leftarrow$ Compute Equation (8)
5 $C_i \leftarrow$ Sort C_i verifying $C_j \leq C_{j+1}$
6 $f_{i,j} \leftarrow$ Sort $f_{i,j}$ verifying $f_{1,j} \leq f_{1,j+1}$ and $f_{i,j} > f_{i+1,j}$
7 **repeat**
`// Select frequency and processor to locate` C_i
8 **repeat**
9 **if** processor j has not been previously selected **then**
10 **repeat**
`//` $\Delta t\left(C_i, f_{j,k}\right)$ `and` $\Delta E\left(C_i, f_{j,k}\right)$ `computed as Equations (5) and (6)`
11 $\Delta\left(C_i, f_{j,k}\right) = a \cdot \Delta t\left(C_i, f_{j,k}\right) + b \cdot \Delta E\left(C_i, f_{j,k}\right)$
12 **until** all $k = FL$ frequency levels are considered;
13 **end**
14 **until** all $j = N_P$ processors are evaluated;
15 $S \leftarrow$ Get the frequency level of processor, j, for which is obtained the minimum value of $\Delta\left(C_i, f_{j,k}\right)$
16 Mark processor j as selected
17 **until** all $i = N_P$ processors are evaluated;
18 **return** S

End

In Algorithm 1, it is considered that the number of available processors, N_P, coincides with the number of tasks, N_T. Nevertheless, it is also possible to use the procedure in case of having more tasks than processors. Thus, if we have N_T tasks and N_P processors with $N_T > N_P$, the $N_T - N_P$ tasks remaining not allocated after allocating the first N_P tasks to the N_P processors can be successively allocated once a processor j completes the task allocated to it. This can be done by evaluating the cost function $\Delta(C_s, f_{j,k})(k = 1, ..., FL)$ for all $N_T - N_P$ remaining tasks, C_s, and the frequencies at which processor j could run. The task for which the lowest value of $\Delta(C_s, f_{j,k})$ is obtained is the one allocated to processor j.

4 | A MULTI-OBJECTIVE SCHEDULING APPROACH FOR CPU-GPU PLATFORM

This section deals with the case of a platform including only two different kinds of processors among which is necessary to distribute a set of tasks with the same workload, allowing a specific way for applying the proposed multi-objective cost function $\Delta(C_i, f_{j,k})$. Although it seems to be a very specific situation, many real applications tackled by evolutionary metaheuristics and executed in parallel on platforms including CPU and GPU cores correspond to this profile (9). Indeed, heterogeneous platforms with nodes including several superscalar multicore microprocessors (CPU cores) and GPUs (although other accelerators such as FPGAs, vector units, etc. are also possible) constitute the present architectural trend to cope with energy efficiency while maintaining performance increments. As it has been briefly introduced in Section 2, the codes to be analyzed evolve a population of individuals distributed among the available CPU and GPU cores of the node to compute their fitness. In the GPU architectures here considered, the GPU cores are grouped defining several multi-threaded processors that, for example in NVIDIA GPUs, are

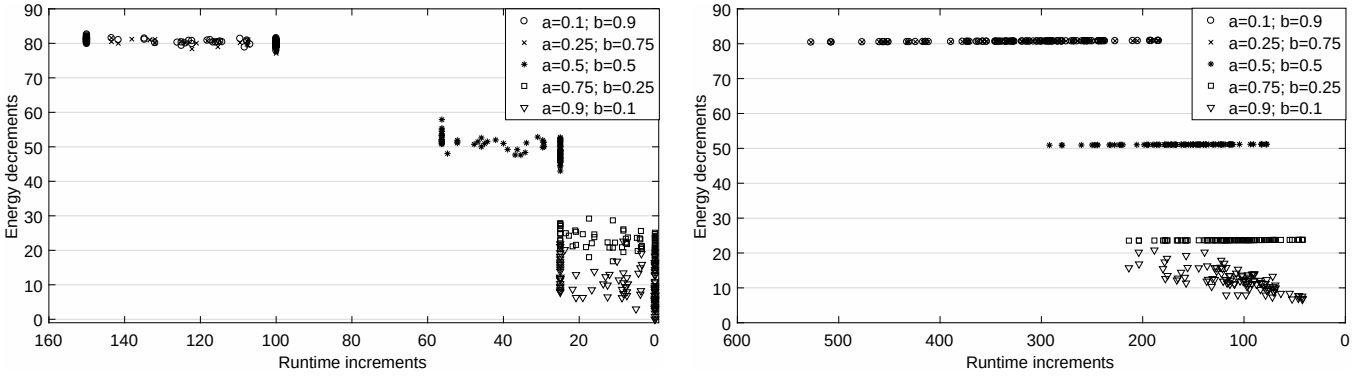


FIGURE 2 Increments in runtime (higher positive values mean worse runtime) and decrements in energy consumption (higher positive values mean less energy consumed) with respect to two scheduling procedures taken as references: (a) results with respect to random allocated tasks; (b) results with respect to minimum runtime scheduling

called Streaming Multiprocessors (SMXs). The fitness evaluation of a given individual is allocated either to a superscalar CPU core or to an SMX. The GPU cores included in a given SMX also allow the parallel fitness evaluation of the corresponding allocated individual by taking advantage of the corresponding data parallelism. Whenever the fitness evaluation of the individuals of the population corresponds to the most important part of the computational cost of the evolutionary algorithm, the workload scheduling is reduced to determine the rate, x, of individuals allocated to the GPU cores and the rate $1 - x$ of them allocated to the CPU cores. Thus, a model for the runtime is:

$$t = g \cdot \left(N \cdot t_{master} + max \left(\left\lceil \frac{x \cdot N}{P_{GPU}} \right\rceil \cdot t_{GPU}, \left\lceil \frac{(1-x) \cdot N}{P_{CPU}} \right\rceil \cdot t_{CPU} \right) \right) \tag{9}$$

where g is the number of generations, N is the number of individuals in the population, and P_{CPU} and P_{GPU} are, respectively, the CPU cores and GPU SMXs in the platform. It is considered that the $x \cdot N$ individuals allocated to the GPU cores are equally distributed among the GPU cores, and the $(1 - x) \cdot N$ individuals are equally distributed among the CPU cores. The parameter t_{master} corresponds to the time required by one of the cores to process the master task $T0$ (Figure 1) for a given iteration, while parameters t_{CPU} and t_{GPU} are, respectively, the time required by the CPU cores and the GPU cores to evaluate one individual. The parameters t_{master}, t_{CPU} and t_{GPU} can be also expressed as a function of the corresponding workload and the frequency of the corresponding processor as $t_{master} = \frac{W_{master}}{F_{CPU}}$, $t_{CPU} = \frac{W_{CPU}}{F_{CPU}}$, and $t_{GPU} = \frac{W_{GPU}}{F_{GPU}}$, where F_{CPU} and F_{GPU} are the frequencies of, respectively, the CPU and GPU cores, and W_{master}, W_{CPU}, and W_{GPU} are respectively, estimations of the cycles of the workloads of $T0$, the evaluation of an individual in the CPU, and the evaluation of an individual in the GPU. This way, the execution time can be modeled as:

$$t = g \cdot \left(N \cdot \frac{W_{master}}{F_{CPU}} + max \left(\left\lceil \frac{x \cdot N}{P_{GPU}} \right\rceil \cdot \frac{W_{GPU}}{F_{GPU}}, \left\lceil \frac{(1-x) \cdot N}{P_{CPU}} \right\rceil \cdot \frac{W_{CPU}}{F_{CPU}} \right) \right) \tag{10}$$

Some considerations have to be done with regard to model of Equation (10). It has been supposed that the times t_{CPU} and t_{GPU} required to evaluate one individual are the same for all individuals in the CPU cores (t_{CPU}) or in the GPU SMXs (t_{GPU}). This circumstance can be also considered very unusual, although it is possible to find useful application showing such behaviour. The EEG feature selection for BCI here considered can be suitably modelled this way as the evaluation of each individual has been done by a fixed number of iterations of the K-means algorithm to qualify the utility of the set of features selected that a given individual in the population codifies. Thus, the data parallelism provided by the GPU SMXs provides similar acceleration to the K-means algorithm for all individuals in the population as has been demonstrated in our previous paper (9). In our problem, it is possible to fit two linear regressions considering the values of the load distribution, x, verifying:

$$\left\lceil \frac{x \cdot N}{P_{GPU}} \right\rceil \cdot \frac{W_{GPU}}{F_{GPU}} \leq \left\lceil \frac{(1-x) \cdot N}{P_{CPU}} \right\rceil \cdot \frac{W_{CPU}}{F_{CPU}} \tag{11}$$

$$\left\lceil \frac{x \cdot N}{P_{GPU}} \right\rceil \cdot \frac{W_{GPU}}{F_{GPU}} > \left\lceil \frac{(1-x) \cdot N}{P_{CPU}} \right\rceil \cdot \frac{W_{CPU}}{F_{CPU}} \tag{12}$$

The corresponding expressions obtained are the next one:

$$T_{left} = T_{left\_0} + T_{left\_1} \cdot \left\lceil \frac{(1-x)\cdot N}{P_{CPU}} \right\rceil \tag{13}$$

where T_{left} stands for the running time in case of low values of x (more workload is allocated to the CPU cores than to the GPU ones), and thus the GPU ends its workload before, and:

$$T_{right} = T_{right\_0} + T_{right\_1} \cdot \left\lceil \frac{x \cdot N}{P_{GPU}} \right\rceil \tag{14}$$

being T_{right} the running time for high values of x, where the CPU cores finish their workload before. Thus, for the given platform (the values of N, P_{CPU}, P_{GPU}, F_{CPU} and F_{GPU} are known), from the known experimental values of T_{left} for $x = 0$, T_{right} for $x = 1.0$, and T_{left} for $x = 0.1$ (or T_{right} for $x = 0.9$) it is possible to determine the values of t_{CPU}, t_{GPU}, and t_{master} (or W_{CPU}, W_{GPU}, and W_{master} as F_{CPU} and F_{GPU} are known). An approximate model for energy consumption is described in what follows. Given a GPU including P_{GPU} cores running at frequency F_{GPU}, the energy consumed by the evaluation of its individuals, distributed among the P_{GPU} cores can be expressed as the product of the instantaneous power consumed by the cores and the corresponding running time, plus the energy consumed by the cores while they are idle. This way, in each generation, g, the energy consumed by the CPU and the GPU can be given as Equation (15) shows:

$$\begin{aligned} E_{CPU} &= Pow_{CPU} \cdot \left\lfloor \frac{(1-x)\cdot N}{P_{CPU}} \right\rfloor \cdot t_{CPU} + \frac{Pow_{CPU}}{P_{CPU}} \cdot \left((1-x)\cdot N - \left\lfloor \frac{(1-x)\cdot N}{P_{CPU}} \right\rfloor \cdot P_{CPU} \right) \cdot t_{CPU} + E^{idle}_{CPU} \\ &= Pow_{CPU} \cdot \frac{(1-x)\cdot N}{P_{CPU}} \cdot t_{CPU} + E^{idle}_{CPU} \\ E_{GPU} &= Pow_{GPU} \cdot \left\lfloor \frac{x \cdot N}{P_{GPU}} \right\rfloor \cdot t_{GPU} + \frac{Pow_{GPU}}{P_{GPU}} \left(x \cdot N - \left\lfloor \frac{x \cdot N}{P_{GPU}} \right\rfloor \cdot P_{GPU} \right) \cdot t_{GPU} + E^{idle}_{GPU} \\ &= Pow_{GPU} \cdot \frac{x \cdot N}{P_{GPU}} \cdot t_{GPU} + E^{idle}_{GPU} \end{aligned} \tag{15}$$

where Pow_{GPU} is the instantaneous power consumed when the P_{GPU} cores of the GPU are evaluating individuals, and E^{idle}_{GPU} is the energy consumed by the idle cores. Pow_{CPU}, P_{CPU}, and E^{idle}_{CPU} are the analogous terms for CPU. The rationale for the previous Equation (15) can be understood from Figure 3 that refers to the GPU but can be also applied to CPU cores (by also substituting x by $1-x$). In the figure, the number of squares in the horizontal dimension (P_{GPU}) corresponds to the number of SMXs in the GPU while the number of squares in the vertical dimension corresponds to the times that the GPU SMXs work in parallel once the $x \cdot N$ individuals are distributed among them. In the last iteration not all available SMXs possibly work. Thus, the grey squares indicate that the SMX works and consumes a $\frac{Pow_{GPU}}{P_{GPU}}$ of power and the white squares correspond to the power consumed while the SMX is idle, $\frac{Pow^{idle}_{GPU}}{P_{GPU}}$. Taking into account the previous expressions, the energy consumed across the generations executed by the algorithm can be given as:

$$\begin{aligned} E &= g \cdot \left(E_{GPU} + E_{CPU} + \epsilon \right) \\ &= g \cdot \left(Pow_{GPU} \cdot \frac{x \cdot N}{P_{GPU}} \cdot t_{GPU} + Pow_{CPU} \cdot \frac{(1-x)\cdot N}{P_{CPU}} \cdot t_{CPU} + E^{idle}_{CPU} + E^{idle}_{GPU} + \epsilon \right) \\ &= g \cdot \left(Pow_{GPU} \cdot \frac{x \cdot N}{P_{GPU}} \cdot \left(\frac{W_{GPU}}{F_{GPU}} \right) + Pow_{CPU} \cdot \frac{(1-x)\cdot N}{P_{CPU}} \cdot \left(\frac{W_{CPU}}{F_{CPU}} \right) + E^{idle}_{CPU} + E^{idle}_{GPU} + \epsilon \right) \end{aligned} \tag{16}$$

The third term of Equation (16), ϵ, corresponds to the energy consumed by task $T0$ and the other elements of the platform (memory, buses, I/O, etc.). Models for E^{idle}_{CPU} and E^{idle}_{GPU} can be also obtained in terms of parameters of the platform and the workload distribution as follows:

$$\begin{aligned} E^{idle}_{CPU} &= Pow^{idle}_{CPU} \cdot \left(\left(1 - \frac{(1-x)\cdot N}{P_{CPU}} + \left\lfloor \frac{(1-x)\cdot N}{P_{CPU}} \right\rfloor \right) \cdot \left(\frac{W_{CPU}}{F_{CPU}} \right) \right) \\ E^{idle}_{GPU} &= Pow^{idle}_{GPU} \cdot \left(1 - \frac{x \cdot N}{P_{GPU}} + \left\lfloor \frac{x \cdot N}{P_{GPU}} \right\rfloor \right) \cdot \left(\frac{W_{GPU}}{F_{GPU}} \right) \end{aligned} \tag{17}$$

The parameters of the models for time and energy consumption from Equations (10) to (17) could be determined from experiments considering different distribution rates, x and $1-x$, number of individuals, N, and number of generations, g, given the characteristics of the CPU-GPU platform in terms of the number of processors, P_{CPU} and P_{GPU}, and their operating frequencies

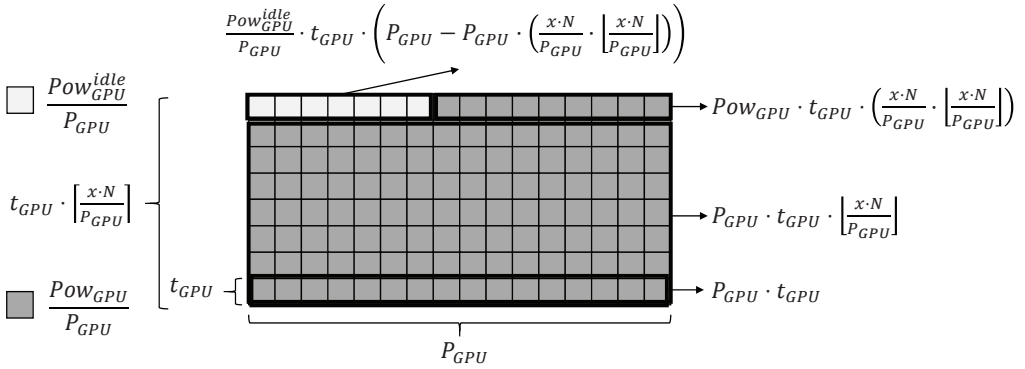


FIGURE 3 Scheme of the energy expressions corresponding to GPU

F_{CPU} and F_{GPU}. By fitting Equation (10) with the experimental time measures, it would be possible to obtain the parameters W_{master}, W_{CPU}, and W_{GPU}. Once these values are substituted in Equations (16) to (17), the experimental values of energy consumption can be used to determine Pow_{CPU}, Pow_{GPU}, Pow_{CPU}^{idle}, and Pow_{GPU}^{idle}, and the shape of ϵ after fitting the model for energy consumption of Equation (16). Thus, we consider a linear regression model that, according to Equations (16) to (17) presents the following terms:

$$E = A_0 + A_1 \cdot \left(\frac{x \cdot N}{P_{GPU}}\right) + A_2 \cdot \left\lfloor \frac{x \cdot N}{P_{GPU}} \right\rfloor + A_3 \cdot \left\lfloor \frac{(1-x) \cdot N}{P_{CPU}} \right\rfloor + \epsilon \tag{18}$$

where coefficients A_0, A_1, A_2, and A_3 can be related with the parameters of the model in Equations (16) to (17) as:

$$A_0 = g \cdot \left(Pow_{GPU}^{idle} \cdot \left(\frac{W_{GPU}}{F_{GPU}}\right) + Pow_{CPU}^{idle} \cdot \left(\frac{W_{CPU}}{F_{CPU}}\right) \cdot \left(1 - \frac{N}{P_{CPU}}\right) + Pow_{CPU} \cdot \left(\frac{N}{P_{CPU}}\right) \cdot \left(\frac{W_{CPU}}{F_{CPU}}\right)\right) \tag{19}$$

$$A_1 = g \cdot \left(\left(\left(Pow_{GPU} - Pow_{GPU}^{idle}\right) \cdot \left(\frac{W_{GPU}}{F_{GPU}}\right)\right) - \left(\left(Pow_{CPU} - Pow_{CPU}^{idle}\right) \cdot \left(\frac{W_{CPU}}{F_{CPU}}\right) \cdot \left(\frac{P_{GPU}}{P_{CPU}}\right)\right)\right) \tag{20}$$

$$A_2 = g \cdot \left(\frac{W_{GPU}}{F_{GPU}}\right) \cdot Pow_{GPU}^{idle} \tag{21}$$

$$A_3 = g \cdot \left(\frac{W_{CPU}}{F_{CPU}}\right) \cdot Pow_{CPU}^{idle} \tag{22}$$

Then, it is possible to build the cost function for conditions of population, generations, frequencies, etc. not previously executed by determining the relative deviations given in Equations (1), and to statically determine the best value of x for a given population of N individuals and operating frequencies to execute the application in a given platform (number of CPU and GPU cores, and the rest of parameters). This approach for static workload scheduling is summarized in Figure 4 and experimentally analyzed in Section 5.

5 | EXPERIMENTAL RESULTS

In this section, we experimentally analyze our approach. The measures of time and consumed energy have been obtained by running an OpenCL (version 1.2) code for the multi-objective feature selection problem corresponding to a BCI task applied to a dataset containing 178 patterns extracted from the data recorded in the BCI Laboratory of the University of Essex. Each pattern is an electroencephalogram (EEG), described by 3,600 features corresponding to 12 features per each of the 20 temporal segments and 15 electrodes (8). The code implements a master-worker parallel multi-objective evolutionary algorithm where each individual codifies a subset of 3,600 features (a feature selection). The fitness function to evaluate the quality of a given feature selection (i.e. the fitness function of the corresponding individual) implies to apply the K-means algorithm to the set of 178 patterns described with the selected features as components (9). In each generation, to evaluate the fitness of the N individuals in the population, the master thread distributes them among the CPU cores and GPU SMXs. The fitness evaluation

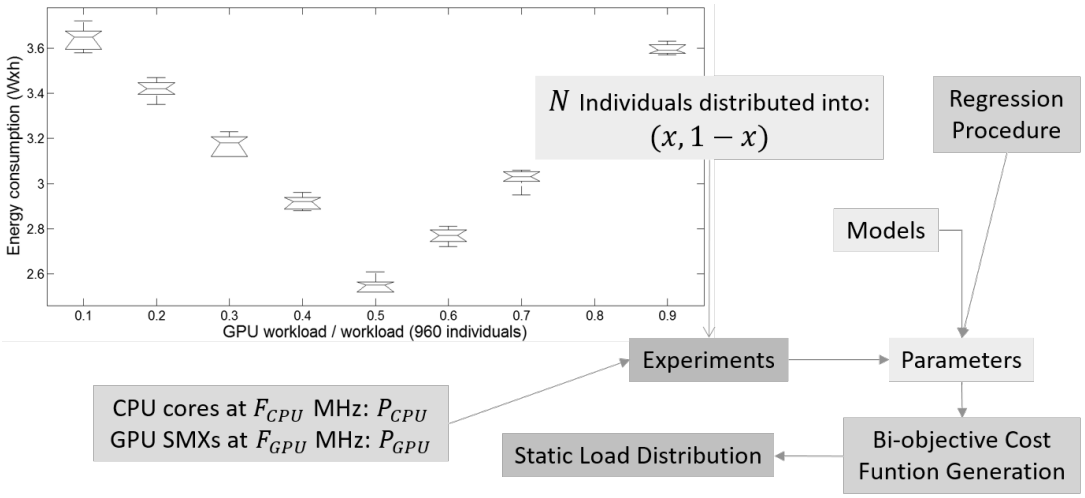


FIGURE 4 Scheme of a static load distribution procedure for CPU-GPU platforms

of each individual is done by one thread in case of the individual is allocated to a CPU core, while if the individual is allocated to the GPU, its fitness is processed in parallel by all threads of one of the Stream Multiprocessors (SMXs) of the GPU.

The code, compiled with GCC 4.8.5, runs on Linux CentOS 6.7, in a cluster node including two Intel Xeon E5-2620 v4 CPUs with eight cores per socket (thus running up to 32 threads per node). The node also includes an NVIDIA Tesla K40m at 755 MHz with 12 GB of global memory, 288 GB/s as maximum memory bandwidth, and 2,880 CUDA cores distributed into 15 SMXs. In our experiments, the CPU cores have executed the threads allocated to them at 1,200 MHz, 1,600 MHz and 2,100 MHz using the *cpupower* shell command. The instantaneous power and energy consumption of the node has been measured by a watt-meter which we have developed based on an *Arduino Mega* card. It provides, in real time, four measures per second for each node in the cluster. We have repeated the experiments five times. Tables 3 and 4 provide the average and standard deviations of the time and energy consumption measures obtained from our experiments. The points in the graphs of Figure 5 provide the time measures and the curves fitted by using our model and multiple linear regression. Furthermore, Figure 6 provides for energy consumption the same information as Figure 5 for running time.

From the experimental measures of the energy consumption, it is clear that these curves do not evolve linearly with x and thus, the ϵ term in Equation (16) is not linear with x as the rest of terms in that equation are linear or, as can be seen from Equations (15), bounded by curves linear with x. The energy consumed by our application depends on the power dissipated by the CPU and GPU cores and the uncore elements of those processors (last level caches, interconnects, etc.), and the power dissipated by the remaining components of the system. It is possible to assume that the power dissipated by the uncore and the system elements is constant once the computer has been executing some workload for a given time (13). It seems that ϵ behaves as a constant term until a given value of x, i.e. it keeps constant until the number of individuals sent to be evaluated by the GPU is larger than a value that corresponds to a volume of data to transfer to the GPU, thus implying more power consumption in the PCIe bus or in the memory transferences. From this value of x, the experimental results seems to correspond to linear increments in ϵ as x grows. It has to be taken into account that, although the time consumed by the memory accesses and bus transferences can be overlapped with the CPU or GPU processing and could not be apparent in time, these elements consume energy that could be apparent in the experimental energy consumption data (if it represents a significant amount with respect to the other energy terms). This way, we have modeled ϵ as:

$$\epsilon = \epsilon_0 \cdot \left(x - x_c\right) \cdot \left\lfloor \frac{x}{x_c} \right\rfloor \tag{23}$$

The value of x_c in Equation (23) can be obtained by two linear regressions. The first one uses experimental results of values of x close to 0, corresponding to workloads much higher in the CPU cores than in the GPU ones (and also values of x lower than x_c), while the second linear regression is applied to values of x close to 1 (and also values higher than x_c), where the GPU is much more loaded than the CPU. The cross point of these two lines estimates the value of x_c. Table 5 provides the values of x estimated for x_c, and for the minima running times, for the different values of N and F_{CPU} considered in our experiments.

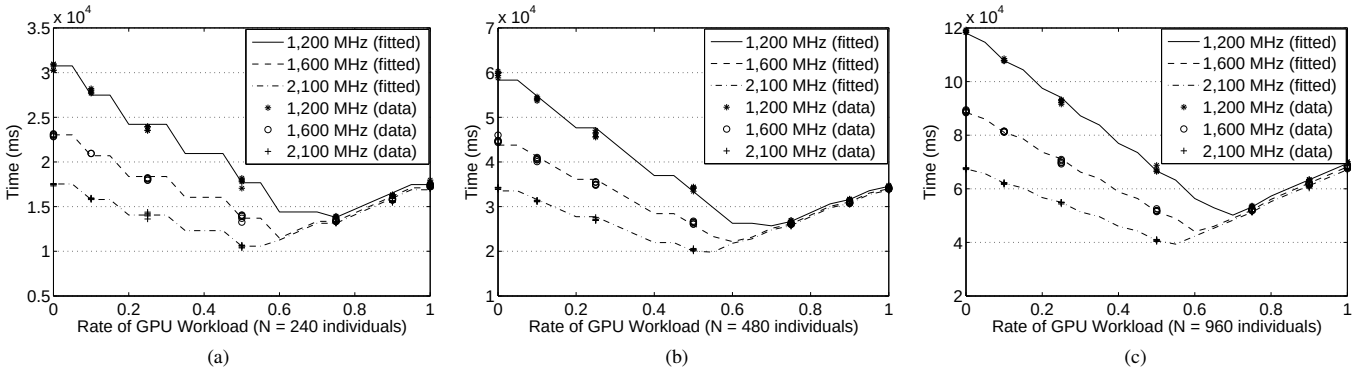


FIGURE 5 Fitted curves for experimental running times with $N = 240$ (a), 480 (b), 960 (c) individuals and $F_{CPU} = 1,200$, 1,600, 2,100 MHz and model defined by Equation (10)

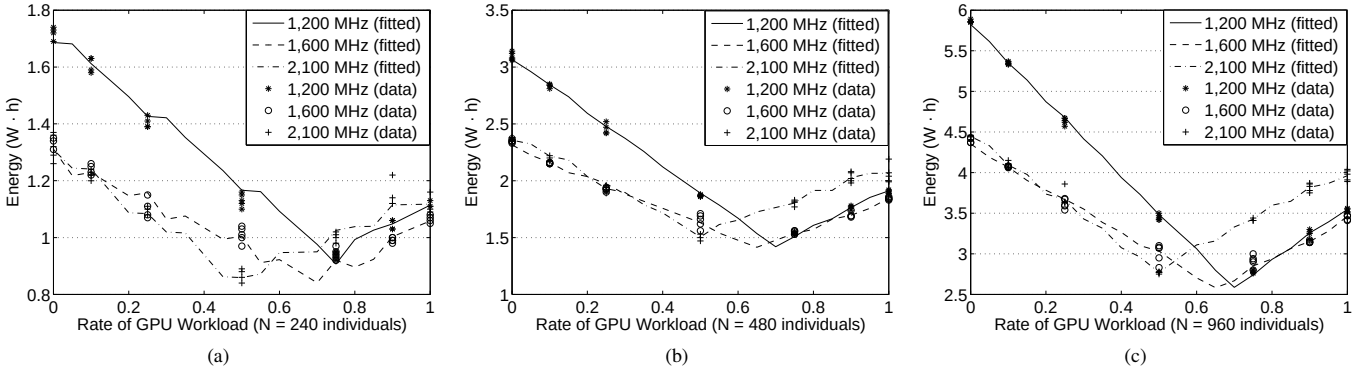
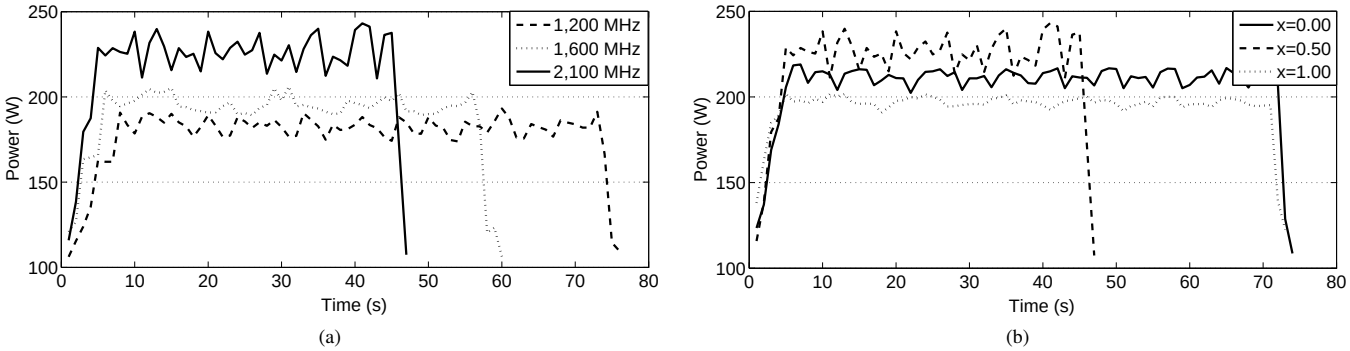


FIGURE 6 Fitted curves for experimental consumed energies with $N = 240$ (a), 480 (b), 960 (c) individuals and $F_{CPU} = 1,200$, 1,600, 2,100 MHz and model defined by Equation (16)

Table 6 shows the values of the R^2-statistics, the F-statistics and its p-values, and the error standard deviations corresponding to the fitted running time and energy consumption models for different values of population, N, and CPU core frequencies, F_{CPU} (Figures 5 and 6). The values shown for R^2-statistic (closed to 1 in all alternatives), F-statistic and its p-value (very low p-values compared with their corresponding F-statistic values), and error standard deviations demonstrate that the proposed time and energy models are statistically significant. Indeed, seeing Figures 5 (a) to 5 (c), and Figures 6 (a) to 6 (c) the accuracy of the fitted curves is acceptable. In particular, the minima of the fitted curves correspond to the ones experimentally observed.

Table 7 provides the parameters of time and energy models described in Equations (10), (16), (18), and (23) once they have been fitted to the experimental data as it is shown in Figures 5 and 6 . The parameters W_{master}, W_{CPU}, and W_{GPU} can be determined from Equations (10) to (14), corresponding to the running time model fitted to the experimental results obtained, for example, in the case of $N = 240$ individuals, $g = 50$, and $F_{CPU} = 1,200$ MHz. The values obtained ($W_{master} = 0.46 \cdot 10^6$, $W_{CPU} = 78.48 \cdot 10^6$ and $W_{GPU} = 13.69 \cdot 10^6$ cycles) allow the specification of Equation (10) for our platform and application. Thus, it makes possible to predict the curves for the values of F_{CPU} (1,200, 1,600 and 2,100 MHz), and N (240, 480 and 960 individuals), used in our experiments. The values provided in Table 7 for the parameters W_{CPU} and W_{GPU} of the fitted models respectively present standard deviations of only 5.2% and 3.5% with respect to their respective mean values. This circumstance seems to corroborate our supposition in the models of Section 4 about the use of a similar number of cycles required to evaluate the fitness for all individuals of the population in an SMX of the GPU (W_{GPU}) or in a CPU core (W_{CPU}). Moreover, $W_{CPU} > W_{GPU}$, which is coherent with the data parallel fitness evaluation of the individuals allocated to GPU SMXs.

Moreover, from the experimental values of the energy consumed by the node after executing the code corresponding to the given values for N, g, F_{GPU}, and F_{CPU} (and once W_{master}, W_{CPU}, and W_{GPU} have been determined from the experimental time



(a) (b)

FIGURE 7 Temporal evolution of the instantaneous power: (a) for $x = 0.5$ and different values of frequency in the CPU cores; and (b) for a frequency of 2,100 MHz in the CPU cores and different values of x (rate of workload allocated to the GPU) and $N = 960$ individuals

measures as has been indicated before), it is possible to determine the parameters Pow_{CPU}, Pow_{GPU}, Pow^{idle}_{CPU}, Pow^{idle}_{GPU}, and ϵ_0 from the multiple linear regression of Equation (18) where the term ϵ is substituted by Equation (23). Equations (19) to (22) provide the values for Pow_{CPU}, Pow_{GPU}, Pow^{idle}_{CPU} and Pow^{idle}_{GPU} from the coefficients A_0 to A_3 of the linear regression.

As it is shown in Table 7 , the parameters Pow_{CPU}, Pow_{GPU}, Pow^{idle}_{CPU}, Pow^{idle}_{GPU}, and ϵ_0, depend on the operating frequency of the CPU cores. The values of Pow_{CPU} are similar for 1,200 and 1,600 MHz but are larger for 2,100 MHz, for all considered population sizes (240, 480, and 960). With respect to Pow_{GPU}, also in all considered population sizes, the values are larger for 1,600 MHz than for 1,200 MHz and 2,100 MHz, which present similar values. The values of Pow_{CPU} are between 1.4 and 3.7 times larger than the values of Pow_{GPU}. With respect to the parameters related with the energy consumption term not explained by the CPU and GPU terms (Equation (15)), and defined in Equation (23), the experimental results show (Table 5) that the values of x_c slightly decrease as the CPU core frequency grows and almost do not change with the population size. Given a population size, the values of ϵ_0 decrease from 1,200 MHz to 1,600 MHz, and slightly increases from 1,600 MHz to 2,100 MHz, as shown in Table 7 . Given a CPU frequency, the values of ϵ_0 clearly grow with the population size, N.

Once the parameters of the models are obtained for one of the considered alternatives (for example $N = 240$ individuals and $F_{CPU} = 1,200$ MHz) it is possible to estimate the running time and energy consumption models for other values of N at the same frequency, F_{CPU}. Table 8 provides the mean of the relative prediction errors for different values of N and F_{CPU} when the parameters of their corresponding time and energy models are estimated from regressions applied to the configuration of $N = 240$ individuals and F_{CPU} equal to 1,200, 1,600, and 2,100 MHz, respectively. As Table 8 shows, the mean relative errors are between 12.1% and 24.6% for the energy consumption model, and between 1.4% and 3.1% for the running time model.

Figure 7 shows curves corresponding to the temporal evolution of the instantaneous power. The curves in Figure 7 (a) illustrate the evolution for different operation frequencies in the CPU cores with the same distribution of individuals among CPU and GPU cores ($x = 0.5$) while Figure 7 (b) gives curves for different distributions of individuals at the same operating frequency in the CPU cores ($F_{CPU} = 2,100$ MHz). From Figure 7 (a), it is clear that the values of the instantaneous power grow with the operating frequencies. Indeed, the values of the instantaneous power for 1,200 and 1,600 MHz are nearer than the ones corresponding to 1,600 and 2,100 MHz. Figure 7 (b) shows that the values of the instantaneous power also change with the rate of individuals allocated to the CPU cores. The curve with the lowest values of instantaneous power corresponds to the situation in which all individuals are allocated to the GPU. With respect to the other two curves, the one with larger power values corresponds to $x = 0.5$. Although only half of the population is allocated to the CPU cores, the power consumed by the elements of the node required to communicate the CPU core of the master thread and the GPU could explain this behaviour.

Figure 8 shows the shape of the cost function $\Delta = a \cdot \Delta t + b \cdot \Delta E$ for $N = 240$ individuals, $F_{CPU} = 1,600$ MHz, and different values for parameters a and b. Depending on these values, the minimum of the cost function corresponds to a minimum in the energy consumption ($x = 0.75$), in the running time ($x = 0.60$), or represents a trade-off between time and energy.

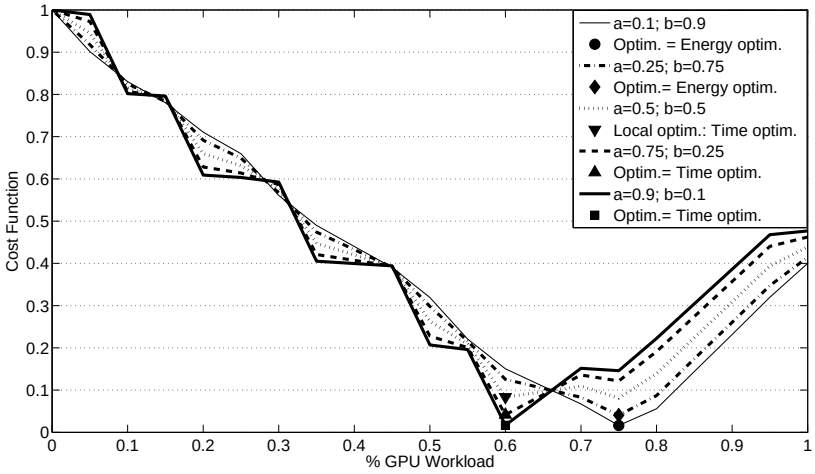


FIGURE 8 Cost function for different values of parameters a and b, for $N = 960$ individuals and $F_{CPU} = 1,600$ MHz

6 | RELATED WORK

In (14), an overview on energy efficiency on clusters is provided. In that paper, the different technologies for energy efficiency are classified into two main groups, the so-called Static Power Management (SPM) techniques, and the Dynamic Power Management (DPM) ones. The SPM techniques are based on the use of low-power components while the DPM techniques use software approaches and power-scalable components. Moreover, the DPM techniques can be also classified according to two alternatives: i) those based on the dynamical adjustment of power consumption by taking advantage of power-scalable components and ii) the energy-aware load balancing techniques.

Several approaches have been reported on scheduling procedures that take into account not only the running time but also the energy consumption of the program, and can be considered as examples of alternative i) in DPM techniques. Most of them are based on Dynamic Voltage Scaling (DVS), a technique that allows dynamic scaling of the CPU voltage to reduce energy consumption. This way, the time spent by lighter tasks waiting for heavy tasks to finish can be reduced and the required levels of performance could be still satisfied. Paper (1) describes a first phase performing a priority based task ordering and scheduling, followed by a second phase where integer programming is used to optimize voltage scaling. In (2), the efficient Decisive Path Scheduling (DPS) algorithm (15) is combined with DVS to minimize both time and energy computation. In the proposed procedure (15), DPS is firstly applied to the corresponding task graph to obtain low finish runtimes and, after that, DVS is applied during slack times to reduce energy consumption while the computing time achieved by the schedule algorithm is maintained. Results obtained by simulation in (2) show average energy consumption reduction of about 40% over DPS.

Paper (3) proposes two different cost functions, based on different approximations for energy measurement, to tackle scheduling on processors with DVS, and reach a trade-off between completion time and energy consumption in precedence-constrained parallel applications. The simulation results provided show that schedules generated by previous *energy-unconscious* scheduling heuristics consume among 16 and 51 percent more energy than their alternatives. In (4), it is described a two-level method for scheduling large workloads to reduce the energy consumption while the quality of service is maximized in a data center with heterogeneous sets of clusters of multi-core processors. The approach is based on the definition of a multi-objective problem for energy-efficient scheduling of workflows in data centers. The experimental results obtained on more than 100,000 workflows generated with the tool described in (16) show that the best performing schedulers defined in (4) achieve improvements up to 46.8% in makespan and up to 29.0% in energy consumption with respect to a typical round-robin strategy. In (17), an approach for scheduling multi-core heterogeneous grid systems that includes the makespan and the minimization of energy consumption as objectives is provided. It uses data on computing capacity and energy consumption from the information provided by vendors and supposes that the tasks to be distributed among the available cores have neither deadlines nor precedence constraints.

The Heterogeneous Energy-aware Race to Halt (H-EARtH) algorithm described in (18), considers the availability of heterogeneous platforms and DVFS capabilities, to provide a runtime procedure that determines the best core and its corresponding voltage and frequency values to optimize energy consumption. In general, the previously cited papers on energy-aware scheduling procedures show relevant improvements in energy consumption. Nevertheless, in the present state of this researching line

it is difficult to provide some fair comparisons to conclude what is the best strategy. Moreover, the performance improvements achieved by the different approaches depend on the characteristics of the applications, and the most part of the experimental results have been obtained either for workloads randomly generated or corresponding to specific task graph descriptions in which the computational costs of the different tasks are supposed to be known. Our procedure described in Section 3 can be applied to master-worker applications where the time of workload allocated to the master is almost zero. It has a computational complexity proportional to the product of number of tasks, processors and available operation frequencies, and provides energy improvements (Figure 2) of up to 80% with increments in the running time (specially in case of comparison with a scheduling procedure that provides the minimum running time), and up to 20% without increments in the running time (in case of comparison with randomly allocated tasks).

An effective energy-aware online or runtime scheduler requires an accurate prediction of the effects of different voltage and frequency levels in different phases of the application (14), whose computational costs are difficult to know in advance. Nevertheless, besides the procedures requiring programmer-exposed DVFS strategies for runtime power management, it is possible to take into account energy consumption and time optimization principles in platforms that do not allow the user to access and control the DVFS alternatives online (or is so costly and should be avoided). Thus, a black-box scheduling approach is proposed in (5), based on an offline power model and an online workload modeling. Papers (19, 20) deal with the determination of power and energy consumption models either by running micro-benchmarks (19) or through the evaluation of energy consumption of the platform components (20). In (13), the modeling of power consumption in codes for sparse linear systems is shown, along with the analysis of the different power-saving modes of CPU cores, to define energy-aware strategies for the corresponding runtime. In this same line, the energy consumption characterization by applying multiple linear regression models is proposed in (12). We have also applied a multiple linear regression model to define a multi-objective cost function to optimize, with respect to both time and energy consumption, the workload distribution among the processors in the platform.

As energy consumption and running time are competing objectives, a multi-objective (more specifically a bi-objective) approach is required to tackle the development of an energy-aware scheduling problem. Indeed (4) proposes as future work the use of multi-objective evolutionary algorithms to learn about the trade-offs evaluated by the two-level schedulers described in the paper. Nevertheless, a scheduling algorithm built on a Pareto-based multi-objective evolutionary algorithm would require a large computing time along with a strategy to select among the different alternatives included in the obtained the Pareto front. The multi-objective approach here described, based on a bi-objective cost function, requires to assign weights to the energy and time objectives, as it is proposed in (21), where a previously defined weighted energy-delay product corresponding to the desired trade-off among both objectives avoids the selection of one of the alternatives included in the Pareto front once an approach to it has been determined. In this paper, we also propose a cost function that comprises the two goals of energy consumption and runtime although not through an energy-delay product but a weighted sum.

Thus, this paper also illustrates how the approach based on a multi-objective cost function can be applied to both DPM alternatives as our bi-objective cost function can be used by a scheduling procedure in a platform including power-scalable components, and by a load balancing procedure. Indeed, the paper shows how it is possible to define a trade-off between energy consumption and running time to determine the most suitable workload distribution for a heterogeneous CPU-GPU platform.

With respect to the energy consumption efficiency of hybrid CPU-GPU platforms, some results on this topic have been provided by papers (22, 23). For example, (22) provides analytical models to get insight into performance gains and energy consumption in different CPU-GPU platforms and concludes that greater parallelism allows opportunities for energy-saving and encourages the development of energy-saving parallel applications. In (24), two alternatives are considered for energy efficiency: to determine a workload distribution among CPU and GPU to reach that both sides finish at the same time, and to coordinately throttle the GPU frequencies and memory according to their utilization. The paper (24) also points out the need for an approach that takes into account both CPU and GPU architectures to minimize energy consumption as, although GPUs have better energy efficiency than CPUs, allocating all workload to the GPU is not necessarily the most energy efficient approach because energy is the product of power and time, and a workload distribution among CPU and GPU cores that allows similar computing times in the different cores could be the most energy efficient alternative. In the present paper, we have also shown that suitable workload distributions among the CPU and GPU cores are able to optimize energy consumption and running time, although not necessarily the same workload distribution is able to simultaneously optimize both objectives. To cope with this multi-objective problem, we have devised a cost function comprising both energy and runtime objectives. This cost function is built from energy consumption and running time models of the considered application that, following the approach described in (12), have been determined by multiple linear regression of the experimental data.

7 | CONCLUSIONS

This paper proposes a bi-objective cost function comprising information of both runtime and energy consumption of the corresponding workload distribution as $\Delta = a \cdot \Delta t + b \cdot \Delta E$, where the effect of both goals, time and energy, is weighted through parameters, a and b ($a \geq 0$, $b \geq 0$, and $a+b = 1$), that control the trade-off between the two goals. This multi-objective approach avoids the need for selecting one of the non-dominated solutions in the front obtained by a Pareto-based multi-objective optimization. Nevertheless, this cost function based on a multi-objective approach requires the value of the parameters a and b of the cost function, which allows the definition of scheduling procedures aware of runtime and energy once a suitable model is available to predict the energy consumption and runtime of a given workload distribution in the computing platform at hand.

A greedy scheduling procedure based on the proposed bi-objective cost function has been evaluated by simulation in case of DVFS mechanisms would be available in heterogeneous architectures including processors with different power consumption profiles. The simulation experiments which we have accomplished considering a master-worker parallel evolutionary application with a negligible master workload have shown that, by using the adequate combination of parameters a and b, it is possible to control the strength of each component, runtime and energy consumption, of the cost function and get relevant energy-savings without an important increase in the runtime.

From measures of runtime and energy consumed by the CPU-GPU heterogeneous computing node while the application is executed considering different CPU frequencies, it is clear that an adequate workload distribution among CPU cores and GPU SMXs could imply less energy consumption than to allocate all workload to the GPU. As the energy is the product of time and power, lower energy consumptions can be shown, even in the case of using higher CPU frequencies, once the right workload distribution is considered. By multiple linear regression, we have fitted time and energy models to the experimental results with good accuracies and statistical significance.

A lot of researching work still has to be completed. On the one side, it would be very useful to have more accurate models for the heterogeneous configurations of processors in terms of their energy consumption capabilities. The application of the approach proposed here considering measures of consumed energy and runtime in the execution of other real parallel codes and heterogeneous platforms should be also accomplished. The improving of energy-saving approaches and models for the elements in the computing platform different from the CPU and GPU cores should be also accomplished for the application here considered and for others. Moreover, the consideration of different operating GPU frequencies could contribute to complete the analysis of the applicability domain for the proposed models.

ACKNOWLEDGEMENTS

Work funded by project TIN2015-67020-P (Spanish "Ministerio de Economía y Competitividad" and ERDF funds). We would like to thank the BCI laboratory of the University of Essex, especially prof. John Q. Gan, for allowing us to use their databases. We also thank the reviewers for their useful comments and suggestions.

References

[1] Zhang Y., Hu X., Chen D.Z.. Task Scheduling and Voltage Selection for Energy Minimization. In: DAC'2002:183–188ACM; 2002; New Orleans, Louisiana, USA.

[2] Baskiyar S., Abdel-Kader R.. Energy aware DAG scheduling on heterogeneous systems. *Cluster Computing*. 2010;13(4):373–383.

[3] Lee Y.C., Zomaya A.Y.. Energy Conscious Scheduling for Distributed Computing Systems Under Different Operating Conditions. *IEEE Transactions on Parallel and Distributed Systems*. 2011;22(8):1374–1381.

[4] Dorronsoro B., Nesmachnow S., Taheri J., Zomaya A.Y., Talbi E-G., Bouvry P.. A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems. *Sustainable Computing: Informatics and Systems*. 2014;4(4):252–261.

[5] Barik R., Farooqui N., Lewis B.T., Hu C., Shpeisman T.. A black-box approach to energy-aware scheduling on integrated CPU-GPU systems. In: CGO'2016:70–81ACM; 2016; Barcelona, Spain.

[6] Talbi E.G.. *Metaheuristics: From Design to Implementation*. John Wiley & Sons; 2009.

[7] Alba E.. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons; 2005.

[8] Ortega J., Asensio-Cubero J., Gan J.Q., Ortiz A.. Classification of motor imagery tasks for BCI with multiresolution analysis and multiobjective feature selection. *BioMedical Engineering OnLine.* 2016;15(1):73.

[9] Escobar J.J., Ortega J., González J., Damas M., Díaz A.F.. Parallel high-dimensional multi-objective feature selection for EEG classification with dynamic workload balancing on CPU-GPU. *Cluster Computing.* 2017;20(3):1881–1897.

[10] Free Software Foundation . GNU Gprof Documentation https://ftp.gnu.org/pub/old-gnu/Manuals/gprof{-}2.9.1/html_node/gprof_toc.htmlAccessed: 2017-02-10.

[11] Deb K.. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons; 2001.

[12] De Sensi D.. Predicting Performance and Power Consumption of Parallel Applications. In: PDP'2016:200-207IEEE; 2016; Heraklion Crete, Greece.

[13] Aliaga J.I., Barreda M., Dolz M.F., Martín A.F., Mayo R., Quintana-Ortí E.S.. Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems. *Cluster Computing.* 2014;17(4):1335–1348.

[14] Valentini G.L., Lassonde W., Khan S.U., et al. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing.* 2013;16(1):3–15.

[15] Park G., Shirazi B., Marquis J., Choo H.. Decisive path scheduling: a new list scheduling method. In: ICPP'1997:472–480IEEE; 1997; Bloomington, IL, USA.

[16] Taheri J., Zomaya A.Y., Khan S.U.. *Grid simulation tools for Job Scheduling and Datafile Replication in Scalable Computing and Communications: Theory and Practice*. Wiley; 2013.

[17] Nesmachnow S., Dorronsoro B., Pecero J.E., Bouvry P.. Energy-Aware Scheduling on Multicore Heterogeneous Grid Computing Systems. *Journal of Grid Computing.* 2013;11(4):653–680.

[18] Rotem E., Weiser U.C., Mendelson A., Ginosar R., Weissmann E., Aizik Y.. H-EARtH: Heterogeneous Multicore Platform Energy Management. *IEEE Computer Magazine.* 2016;49(10):47-55.

[19] Hong S., Kim H.. An Integrated GPU Power and Performance Model. *SIGARCH Computer Architecture News.* 2010;38(3):280–289.

[20] Ge R., Feng X., Burtscher M., Zong Z.. PEACH: A Model for Performance and Energy Aware Cooperative Hybrid Computing. In: CF'2014:24:1–24:2ACM; 2014; Cagliari, Italy.

[21] Ge R., Feng X., Cameron K.W.. Improvement of Power-Performance Efficiency for High-End Computing. In: IPDPS'2005:233–240IEEE Computer Society; 2005; Denver, Colorado, USA.

[22] Marowka A.. Energy Consumption Modeling for Hybrid Computing. In: Euro-Par'2012:54–64Springer; 2012; Rhodes Island, Greece.

[23] Allen T., Ge R.. Characterizing Power and Performance of GPU Memory Access. In: E2SC'2016:46–53IEEE Press; 2016; Salt Lake City, Utah, USA.

[24] Ma K., Li X., Chen W., Zhang C., Wang X.. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In: ICPP'2012:48-57IEEE; 2012; Pittsburgh, PA, USA.

TABLE 1 Runtime distribution (including time percentages) among the most relevant steps of an evolutionary multi-objective feature selection procedure (8) for different population sizes, N

N	Fitness Evaluation		Non-domination Sorting		Crossover		Rest of Procedures		Total Time
	Time (s)	%	Time (s)	%	Time (s)	%	Time (s)	%	
120	119.19	99.93	0.01	0.01	0.03	0.03	0.04	0.03	119.27
240	236.38	99.92	0.07	0.03	0.09	0.04	0.03	0.01	236.57
480	477.00	99.90	0.14	0.03	0.18	0.04	0.14	0.03	477.46
960	954.85	99.87	0.70	0.07	0.31	0.03	0.29	0.03	956.15
15,000	14,946.12	98.60	196.61	1.30	5.53	0.04	9.87	0.06	15,158.13

TABLE 2 Relative speeds (in %) in the processors with the configuration used in the simulation

%	P1	P2	P3	P4	P5	P6	P7	P8
Configuration	80	80	80	80	100	100	100	100
	64	64	64	64	80	80	80	80
	40	40	40	40	50	50	50	50

TABLE 3 Means and standard deviations of the experimental running times, in seconds (N: Number of individuals in the population; F_{CPU}: Frequencies in the CPU cores; %GPU Workload is the percentage of the population evaluated by the GPU)

N	F_{CPU} (MHz)	% GPU Workload						
		0	10	25	50	75	90	100
240	1,200	30.64 ±0.35	27.93±0.22	23.78±0.22	17.78±0.43	13.86±0.06	16.28±0.04	17.76±0.19
	1,600	22.99 ±0.13	20.95±0.01	18.09±0.12	13.79±0.32	13.44±0.14	15.83±0.09	17.35±0.12
	2,100	17.47±0.10	15.89±0.07	14.05±0.29	10.55±0.13	13.22±0.07	15.63±0.18	17.13±0.12
480	1,200	59.65±0.58	54.13±0.32	46.24±0.63	34.08±0.38	26.70±0.23	31.58±0.11	34.56±0.12
	1,600	44.61±0.16	40.57±0.35	35.14±0.34	26.39±0.28	26.17±0.14	30.93±0.25	34.09±0.17
	2,100	34.15±0.19	31.28±0.15	27.14±0.28	20.33±0.21	25.80±0.17	30.62±0.08	33.76±0.25
960	1,200	118.73±0.41	107.97±0.36	92.56±0.68	67.31±0.91	53.35±0.27	62.95±0.72	69.68±0.36
	1,600	88.88±0.41	81.32±0.15	70.06±0.64	51.89±0.39	52.11±0.22	61.86±0.33	68.31±0.52
	2,100	67.58±0.25	61.99±0.28	54.77±0.27	40.75±0.30	51.36±0.12	60.84±0.42	67.31±0.29

TABLE 4 Means and standard deviations of the experimental energy consumption of the node in $W \cdot h$ (N: Number of individuals in the population; F_{CPU}: Frequencies in the CPU cores; %GPU Workload is the percentage of the population evaluated by the GPU)

N	F_{CPU} (MHz)	% GPU Workload						
		0	10	25	50	75	90	100
240	1,200	1.71±0.02	1.61±0.03	1.40±0.02	1.13±0.02	0.94±0.01	1.05±0.02	1.11±0.01
	1,600	1.34±0.02	1.24±0.02	1.10±0.03	1.01±0.03	0.94±0.02	0.99±0.01	1.06±0.01
	2,100	1.32±0.05	1.22±0.02	1.09±0.01	0.87±0.02	1.00±0.02	1.14±0.04	1.11±0.03
480	1,200	3.09±0.03	2.83±0.02	2.46±0.04	1.87±0.01	1.54±0.01	1.77±0.02	1.90±0.01
	1,600	2.35±0.02	2.15±0.01	1.92±0.02	1.65±0.06	1.55±0.02	1.70±0.02	1.84±0.01
	2,100	2.37±0.01	2.20±0.02	1.93±0.03	1.50±0.02	1.81±0.03	2.03±0.04	2.06±0.08
960	1,200	5.86±0.02	5.35±0.02	4.63±0.04	3.45±0.04	2.79±0.02	3.26±0.05	3.54±0.02
	1,600	4.39±0.03	4.07±0.01	3.62±0.06	3.01±0.12	2.91±0.07	3.15±0.01	3.45±0.03
	2,100	4.44±0.03	4.09±0.04	3.68±0.10	2.77±0.01	3.42±0.01	3.82±0.05	3.97±0.07

TABLE 5 Minima Values (0.05 is the lowest change in the workload considered in our experiments)

N	F_{CPU} (MHz)	$x_c \pm 0.05$	$x\left(t_{min}\right) \pm 0.05$
240	1,200	0.75	0.75
	1,600	0.75	0.60
	2,100	0.50	0.50
480	1,200	0.70	0.70
	1,600	0.65	0.60
	2,100	0.50	0.55
960	1,200	0.70	0.70
	1,600	0.65	0.60
	2,100	0.50	0.55

TABLE 6 Statistical Analysis. $FESD$: Fitting Error of Standard Deviation

N	F_{CPU} (MHz)	R^2-statistic	F-statistic	p-value	$FESD$ (W · h)	R^2-statistic	F-statistic	p-value	$FESD$ (s)
240	1,200	0.996	1662.02	$9.76 \cdot 10^{-35}$	0.014	0.992	2279.00	$2.07 \cdot 10^{-20}$	0.017
	1,600	0.945	128.30	$2.06 \cdot 10^{-18}$	0.029	0.995	3348.98	$6.63 \cdot 10^{-22}$	0.016
	2,100	0.947	133.64	$1.16 \cdot 10^{-18}$	0.028	0.996	4592.69	$3.91 \cdot 10^{-23}$	0.014
480	1,200	0.998	3557.96	$1.12 \cdot 10^{-39}$	0.012	0.985	1222.98	$5.31 \cdot 10^{-18}$	0.019
	1,600	0.977	325.64	$3.03 \cdot 10^{-24}$	0.021	0.982	984.20	$3.64 \cdot 10^{-17}$	0.019
	2,100	0.989	695.50	$4.18 \cdot 10^{-29}$	0.014	0.990	1772.77	$1.95 \cdot 10^{-19}$	0.015
960	1,200	0.999	7610.05	$1.27 \cdot 10^{-44}$	0.009	0.997	6031.97	$3.39 \cdot 10^{-24}$	0.011
	1,600	0.993	1082.51	$5.83 \cdot 10^{-32}$	0.013	0.997	6026.96	$3.42 \cdot 10^{-24}$	0.009
	2,100	0.991	815.53	$3.94 \cdot 10^{-30}$	0.012	0.999	23591.77	$1.61 \cdot 10^{-29}$	0.006

TABLE 7 Parameters of time and energy models described in Equations (10), (16), and (23) fitted by linear regression

N	F_{CPU} (MHz)	W_{CPU} ($\cdot 10^6$ Cycles)	W_{GPU} ($\cdot 10^6$ Cycles)	W_{master} ($\cdot 10^6$ Cycles)	$\frac{Pow_{CPU}}{P_{CPU}}$ (W)	$\frac{Pow_{GPU}}{P_{GPU}}$ (W)	$\frac{Pow^{idle}_{CPU}}{P_{CPU}}$ (W)	$\frac{Pow^{idle}_{GPU}}{P_{GPU}}$ (W)	ϵ_0 (W · h)
240	1,200	78.48	13.79	0.46	7.96	9.62	1.41	2.48	1.82
	1,600	74.62	14.05	0.58	8.23	12.11	1.53	4.46	1.13
	2,100	73.16	14.06	0.63	11.70	6.74	1.37	2.05	1.57
480	1,200	85.46	14.72	0.25	6.55	5.57	1.47	1.50	4.11
	1,600	81.80	14.79	0.36	6.70	7.86	1.53	2.63	2.71
	2,100	81.00	14.87	0.40	9.67	5.60	2.95	2.12	2.88
960	1,200	82.30	15.22	0.38	6.44	4.39	2.78	1.38	8.07
	1,600	79.09	15.13	0.48	6.44	6.17	2.99	2.35	4.82
	2,100	74.96	14.88	0.61	9.52	4.44	3.41	1.43	5.75

TABLE 8 Mean Relative Error, MRE, in time and energy prediction of the case for $N = 240$ individuals, with respect to $N = (480, 960)$ at the same frequency, F_{CPU}

N	F_{CPU} (MHz)	MRE (Energy prediction)	MRE (Time prediction)
		$N = 240$ and $F_{CPU} = 1,200$	
480	1,200	0.166±0.035	0.031±0.026
960		0.223±0.042	0.015±0.011
		$N = 240$ and $F_{CPU} = 1,600$	
480	1,600	0.171±0.035	0.023±0.019
960		0.246±0.049	0.014±0.011
		$N = 240$ and $F_{CPU} = 2,100$	
480	2,100	0.108±0.012	0.020±0.015
960		0.179±0.013	0.016±0.013