# Multi-objective Feature Selection for EEG Classification with Multi-Level Parallelism on Heterogeneous CPU-GPU Clusters

Juan José Escobar*
CITIC, University of Granada
Granada, Spain
jjescobar@ugr.es

Julio Ortega
CITIC, University of Granada
Granada, Spain
jortega@ugr.es

Antonio Francisco Díaz
CITIC, University of Granada
Granada, Spain
afdiaz@ugr.es

Jesús González
CITIC, University of Granada
Granada, Spain
jesusgonzalez@ugr.es

Miguel Damas
CITIC, University of Granada
Granada, Spain
mdamas@ugr.es

## ABSTRACT

The present trend in the development of computer architectures that offer improvements in both performance and energy efficiency has provided clusters with interconnected nodes including multiple multi-core microprocessors and accelerators. In these so-called heterogeneous computers, the applications can take advantage of different parallelism levels according to the characteristics of the architectures in the platform. Thus, the applications should be properly programmed to reach good efficiencies, not only with respect to the achieved speedups but also taking into account the issues related to energy consumption. In this paper we provide a multi-objective evolutionary algorithm for feature selection in electroencephalogram (EEG) classification, which can take advantage of parallelism at multiple levels: among the CPU-GPU nodes interconnected in the cluster (through message-passing), and inside these nodes (through shared-memory thread-level parallelism in the CPU cores, and data-level parallelism and thread-level parallelism in the GPU). The procedure has been experimentally evaluated in performance and energy consumption and shows statistically significant benefits for feature selection: speedups of up to 73 requiring only a 6% of the energy consumed by the sequential code.

## CCS CONCEPTS

• **Computing methodologies** → *Distributed algorithms*; • **Applied computing** → **Bioinformatics**; • **Theory of computation** → **Massively parallel algorithms**;

## KEYWORDS

Parallel Programming, Heterogeneous Platform, Distributed Master-worker Procedure, Energy-aware Computing, Subpopulation-based Genetic Algorithm, EEG Multi-objective Feature Selection

*Correspondence author


*GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan*

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208239

**ACM Reference Format:**
Juan José Escobar, Julio Ortega, Antonio Francisco Díaz, Jesús González, and Miguel Damas. 2018. Multi-objective Feature Selection for EEG Classification with Multi-Level Parallelism on Heterogeneous CPU-GPU Clusters. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan,* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208239

## 1 INTRODUCTION

Evolutionary algorithms constitute useful tools to tackle many compute-intensive problems on classification, clustering, feature selection, and optimization. These problems are usually posed by data mining applications [15, 16] belonging to areas of great social interest, such as bioinformatics, biomedical engineering, or Internet of Things (IoT). In many of these applications, the embedded evolutionary algorithm performs in high-dimensional spaces, thus requiring such amount of runtime that the only way to get solutions with enough quality in an acceptable runtime is to take advantage of the different levels of parallelism available in the present architectures. This way, the implementation of efficient parallel evolutionary algorithms constitutes an exciting and useful researching area. Moreover, as energy-saving has become an important issue in computer science and engineering due to economic and environmental reasons, efficiency not only means good speedups but also optimal energy consumption.

The most common computing servers available today are clusters whose interconnected nodes implement Non-Uniform Memory Access (NUMA) architectures of multi-core microprocessors and accelerators such as Graphics Processing Units (GPUs) or Field-Programming Gate Arrays (FPGAs). Precisely, these heterogeneous architectures constitute the current answer of the technology to the requirements of computational power and energy consumption efficiency [4, 14, 17]. The GPU usually provide massive Data-Level Parallelism (DLP) along with Thread-Level Parallelism (TLP), which can also be used through multi-core microprocessors (or CPU) implementing Instruction-Level Parallelism (ILP). Moreover, besides offering opportunities to execute efficient parallel codes, the heterogeneous architectures which include CPU and GPU cores could also constitute an efficient approach for energy-saving, and papers such as [14] consider the efficient cooperation of CPU and GPU as an important concern to reach exascale performances.

This way, the development of energy-performance efficient codes for heterogeneous CPU-GPU systems needs to address hardware and software issues related with the cooperation among CPU-GPU nodes [22], along with challenges involved in CPU-GPU heterogeneous computing. Among those, we have the size of CPU and GPU memories, the CPU-GPU memory bandwidth limitations, the load balancing among the CPU and GPU cores, the overlapping of data transfer with CPU and GPU computation, and the parallelism profile of the application considered. This paper proposes a new evolutionary multi-objective procedure for feature selection in EEG classification [19]. With respect to our previously proposed parallel procedures on this application [5, 8], that dynamically distribute the evaluation of individuals among both CPU and GPU cores besides taking advantage of the data parallelism available in GPUs, our contribution in this paper deals with a Message Passing Interface (MPI) extension of those parallel heterogeneous CPU-GPU procedures. This way, using multiple CPU-GPU nodes connected in clusters would allow higher speedups. Moreover, and as in [5], we also evaluate the power-performance of the proposed procedure. Despite the relevance of energy-saving issues, apart from [9] that compares the energy consumption of a sequential evolutionary algorithm in different platforms, there are not many papers that analyze parallel evolutionary algorithms according to their energy consumption.

In the paper, after this introduction, a brief presentation of the application on multi-objective feature selection for EEG classification on Brain Computer Interface (BCI) is given in Section 2. Following that, Section 3 describes the here proposed parallel multi-level subpopulation-based procedure for evolutionary multi-objective feature selection, and Section 4 summarizes the previous related work in the area. Finally, the analysis of the experimental work and the conclusions are respectively provided in Sections 5 and 6.

## 2 MULTI-OBJECTIVE FEATURE SELECTION

This paper tackles Multi-Objective Feature Selection (MOFS) in unsupervised classification of patterns characterized by a high number of features. Multi-objective optimization can be applied to data mining applications, and its benefits in both supervised and unsupervised classification have been reported elsewhere [11]. The most relevant features should be selected in order to get good classification performance besides decreasing the computational cost of the classification, among others. Due to the *NP*-hard complexity of this kind of applications, it is necessary to use parallel metaheuristics that are capable of reducing running time and energy consumed by the algorithm. This way, here we propose the use of heterogeneous parallel architectures to implement them.

A multi-objective evolutionary procedure, in our case the NSGA-II algorithm, evolves one or multiple subpopulations of individuals that codify different feature selections (details in [8]). Each individual performs a $K$-means algorithm to evaluate its multi-objective fitness, which is composed by two cost functions, $f_1$ and $f_2$, and defined according to two Clustering Validation Indices (CVIs) [1], which correspond to the minimization and maximization of the intra-cluster and the inter-cluster distances, respectively, as it is shown in Equations (1) and (2), where $|C^t(j)|$ is the number of patterns in the cluster $C^t(j)(j = 1, .., W)$ whose centroid is $K^t(j)$, and $\|P_i - K^t(j)\|$ is the Euclidean distance between the pattern $P_i$ and the centroid $K^t(j)$. Considering the characteristics of $K$-means and the evaluation of the cost functions, $f_1$ and $f_2$, the $K$-means complexity for each individual can be summarized in Equation (3).

$$f_1 = 1 - \left( \sum_{j=1}^{W} \frac{1}{|C^t(j)|} \cdot \left( \sum_{P_i \in C^t(j)} \|P_i - K^t(j)\| \right) \right) \quad (1)$$

$$f_2 = \sum_{j=1}^{W-1} \cdot \left( \sum_{i>j} \|K^t(i) - K^t(j)\| \right) \quad (2)$$

$$T_{K-means} = W \cdot N_P \cdot T_{Dist} + T_W + \left( N_P + W^2 \right) \cdot T_{Dist} \quad (3)$$

being $T_{Dist}$ the computation of the distance between a pattern and a centroid, and $T_W$ the time required to obtain the new centroids, which depends on the number of patterns, $N_P$, and the number of features, $N_F$. The expression $\left(N_P + W^2\right) \cdot T_{Dist}$ estimates the cost to compute the two objective functions, $f_1$ and $f_2$. The previous equations, corresponding to the multi-objective feature selection can be parallelized by a master-worker approach that distributes the evaluation of individuals. In what follows, we present some details about the implementation of this parallel model in CPU-GPU heterogeneous clusters.

## 3 MULTI-LEVEL PARALLELISM FOR MOFS

The proposed parallel procedure for multi-objective feature selection (Algorithms 1 and 2) takes advantage of up to four parallelism levels depending on the OpenCL [12] devices used to evaluate the fitness of the individuals. The main advantage of dividing parallelism into multiple levels (or layers), is that, at a given time, a particular level can be optimized without depending on the implementation of the rest since each level is hierarchized. The following subsections detail the characteristics of each level according to its implementation and optimization. Figure 1 summarizes all procedure steps, starting with the initial distribution of subpopulations in the master node, and ending with the fitness evaluation of each individual in the remaining nodes (workers).

### 3.1 First Level: Distribution of subpopulations by nodes

The first parallelism level corresponds to a master-worker approach which uses MPI, allowing dynamic distribution of the initial subpopulations over the different workers used in the cluster. We employ this approach instead of a static distribution to avoid unbalanced work. Firstly, observing Algorithm 1, the master (MPI process with rank number 0) reads the configuration files, obtains the centroids from dataset *DS*, and initializes the individuals of the subpopulations (lines 2-3). At this point, master and workers (the other MPI processes) are synchronized and are ready to start the MPI section.

This way, the master broadcasts the centroids to all workers (lines 4-6), which are necessary to perform the $K$-means algorithm. Then, the distribution of subpopulations to be evolved and the global migration are repeated as many times as the number of global migrations, $N_{Gm}$, have been defined (lines 7-17). The master asynchronously begins to attend the requests of each worker, dynamically distributing subpopulations until there is no more work to do (lines 9-15). Now, the master proceeds to perform the global

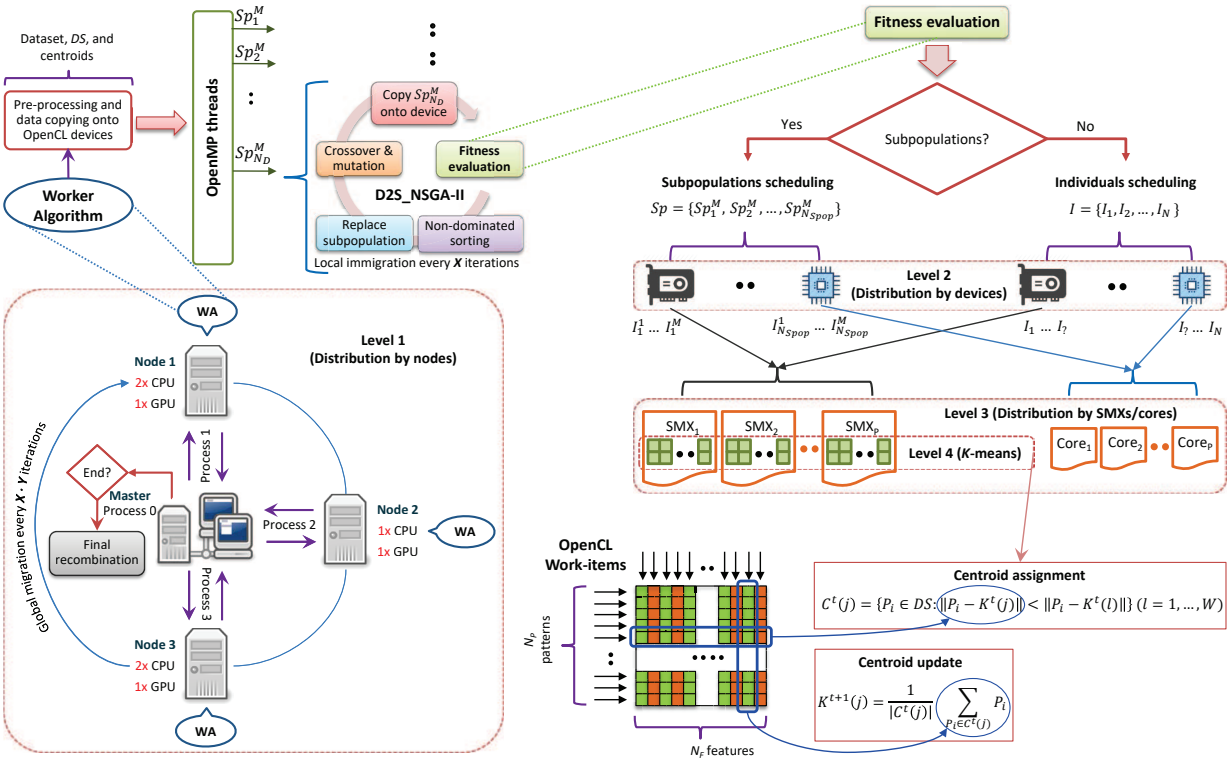


**Figure 1: General scheme of the evolutionary procedure, which provides a multi-level parallelism up to 4 levels depending on the OpenCL devices used to evaluate the fitness of the individuals**

migration (line 16) between all $N_{Spop}$ subpopulations to improve their diversity, increasing the quality of the solutions and trying to avoid the local optima. A global migration implies to build a new set of subpopulations. To define a new subpopulation, the given set of solutions in the subpopulation receives solutions from the rest of subpopulations. More specifically, each subpopulation contributes with half of its solutions in its present Pareto's front at most.

Once all $N_{Spop}$ subpopulations have been evolved, and all $N_{Gm}$ global migrations have been completed, the master sends the signal (*END_SIGNAL*) to the workers to notify that there are no more subpopulations to be evolved and the MPI section has ended (line 18). Moreover, the solutions obtained by the different subpopulations are recombined by the master (lines 19-20) to perform the final subpopulation, which includes the best individuals of each subpopulation belonging to the Pareto's front. Finally, the solution obtained is returned to the main function.

## 3.2 Second Level: Distribution of individuals or subpopulations by OpenCL devices

While the master process schedules the distribution of subpopulations, the workers (Algorithm 2) perform all steps of the evolutionary procedure for each subpopulation. To do that, each worker requests to the master as many subpopulations as $N_D$ OpenCL devices are present in the node (line 6). However, the number of subpopulations received, $Sp_l$, could be lower than $N_D$ if there are not enough subpopulations available to be evolved, as it is shown in line 7. Previously, after starting the MPI section, the OpenCL devices must be initialized, and then copy the necessary data for the evolutionary process, such as the centroids received from the master, $K$, and the $DS$ dataset, as can be seen in lines 2-4.

The evolutionary procedure executed by the worker represents a slight improvement of the parallel multi-objective evolutionary algorithm based on subpopulations, `D2S_NSGAII` (Dynamic Distribution of Subpopulations using NSGA-II), which was already discussed in [5]. In the OpenMP [18] section, as many CPU threads as subpopulations received, $rcv \leq N_D$, are created through the corresponding OpenMP pragma to parallelize the evolutionary steps (lines 9-15), which are repeated according to the required number of subpopulation generations. This way, each subpopulation, $Sp_{l_i}$, is assigned to one of these CPU threads, which manage and execute the evolutionary operators for their corresponding subpopulation (crossover and mutation in line 10, and replacement in lines 12-14). However, the evaluation of individuals (`evaluation` function in line 11) may be executed either on the free CPU cores or on other accelerators, depending on whether the CPU thread associated with $Sp_{l_i}$ manages the CPU or not.

To summarize, this constitutes the second level of parallelism because the subpopulations received by the workers are assigned to the available OpenCL devices, either CPU itself or other devices. However, as the workers will not require more work to the master until their $N_D$ subpopulations have been computed, we think that this strategy may cause load imbalance since the OpenCL devices

**Algorithm 1:** Pseudocode of the distributed master algorithm. The subpopulations are distributed by the master among all worker nodes. Each worker requests the same quantity of subpopulations as OpenCL devices are available

```
1  Function Master(Sp, N_Spop, M, DS)
       Input  : Initial subpopulations, Sp_i; ∀i = 1, ..., N_Spop
       Input  : Number of subpopulations, N_Spop, to evolve
       Input  : Number of individuals in the subpopulation, M
       Input  : Set DS: N_P training patterns of N_F features
       Input  : Number of workers, N_W
       Output : hv, the hypervolume metric
       // W centroids randomly chosen from DS
2      K ← getCentroids(DS)
3      Sp ← initSubpopulations(S_p)
       // Start MPI section
4      repeat
5          K → sendCentroids(K)
6      until K has been sent to all N_W workers;
7      repeat
           // Dynamic distribution of subpopulations
8          RemainingWork ← N_Spop
9          repeat
10             S_R ← The worker requests S_R subpopulations
11             sent ← min(RemainingWork, S_R)
12             S_p → The master sends sent subpopulations
13             RemainingWork ← RemainingWork − sent
14             S_p ← The master receives sent subpopulations
15         until all N_Spop subpopulations are evaluated;
16         Sp ← globalMigration(Sp, N_Spop, M)
17     until all N_Gm global migrations are completed;
       // End MPI section
18     END_SIGNAL → send the signal to all N_W workers
       // Recombination process
19     Sp ← nonDomSorting(Sp, N_Spop · M)
20     S ← Copy the first M individuals of Sp
21     hv ← zitzlerHypervolume(S)
22     return hv
23 End
```

**Algorithm 2:** Pseudocode of the worker algorithm. The received subpopulations from the master are distributed among the OpenCL devices. If only one subpopulation has been received, its individuals are dynamically assigned to all devices to perform the fitness evaluation

```
1  Function Worker(M, DS, N_D)
       Input : Number of individuals in the subpopulation, M
       Input : Dataset DS: N_P training patterns of N_F features
       Input : Number of OpenCL devices, N_D
       // Start MPI section
2      K ← receiveCentroids()
3      D ← initOpenCLDevices(N_D)
4      D ← Copy DS and K to the OpenCL devices
5      repeat
           // As many subpopulations as N_D devices
6          N_D → The worker requests N_D subpopulations
7          Sp_l ← The worker receives rcv subpopulations
8          repeat
               /* Start OpenMP section with rcv CPU
                  threads and N_D OpenCL devices
                  (D2S_NSGA-II algorithm)          */
9              repeat
10                 O_i ← UniformCrossover(Sp_{l_i})
11                 O_i ← evaluation(O_i, M, DS, K, D, N_D)
                   // Replacement process
12                 Aux_i ← Join Sp_i and O_i in one array
13                 Aux_i ← nonDomSorting(Aux_i, size(Aux_i))
14                 Sp_{l_i} ← Copy the first M individuals of Aux_i
15             until the number of generations, g, is reached;
               // End OpenMP section
16             Sp_l ← localMigration(Sp_l, rcv, M)
17         until all N_Lm local migrations are completed;
18         Sp_l → The worker returns rcv subpopulations
19     until the END_SIGNAL is received;
20 End
```

are not homogeneous, being this imbalance even greater when their computing capabilities differ in great magnitude. Normally some devices will finish their work before others, causing idle time for the most powerful devices, and thus, not only an increase in energy consumption but also a reduction in the acceleration of the algorithm. We are aware of its importance, and therefore, in future work, we will study possible improvements that avoid load imbalance, such as a previous analysis of the computing capacity of each device to assign more work to those that are more powerful, or modify the clock frequency of the cores to match the running time of the devices, allowing better performance and energy-saving by eliminating the idle time.

On the other hand, if the worker receives only one subpopulation, $rcv = 1$, and one OpenMP thread is created, the second parallelism level is preserved because the `evaluation` function detects this situation and dynamically distributes individuals of that subpopulation among all OpenCL devices. Thus, two dynamic scheduling alternatives for evaluation of individuals are present. Figure 1 illustrates more clearly this situation. The local migration is carried out in the same way the master performs the global migration, but among subpopulations that belong to the same worker

(line 16). The D2S_NSGAII algorithm and the local migration are repeated as many times as the number of local migrations, $N_{Lm}$, have been established. Once the whole process is finished, the worker proceeds to return to the master the $Sp_l$ subpopulations already evolved (line 18) and waits for the assignment of more work (subpopulations), or the *END_SIGNAL* signal (line 19), implying that all $N_{Gm}$ global migrations have been carried out by the master, and thus, the workers can return.

### 3.3 Third and Fourth Level: Distribution of individuals by SMXs or CPU cores and GPU Data Parallelism

The third and fourth parallelism levels occur within the evaluation function (line 11 of Algorithm 2). Regardless of whether one or multiple subpopulations have to be evaluated, the third level works with individuals by launching OpenCL kernels to perform the fitness evaluation, as was proposed in [6], and optimized and analyzed in [7, 8]. Each device distributes individuals on their computing units, which can be CPU cores or Streaming Multiprocessors (SMXs), in case of GPUs. In both alternatives, just before calling the kernel, the individuals to be evaluated must be transferred to the devices.

As has been said in Section 2, the fitness evaluation is carried out by applying a $K$-means algorithm over each individual. In CPU, since each individual is assigned to one core, $K$-means is sequentially executed within of that core. On the contrary, in GPU, due to each SMX is composed by multiple CUDA cores (work-items in the OpenCL nomenclature), the data parallelism available in $K$-means allows the parallelization of the assignment and updating of centroids (see Figure 1), which constitutes the fourth (and last) parallelism level. We are aware that data parallelism is also possible in the CPU cores taking into account the usually available vectorization instructions. It would allow efficient use of the architecture although, for simplicity, this remains as future work.

## 4 RELATED WORK

A relatively high number of contributions on parallel implementations of evolutionary algorithms considering CPU-GPU platforms can be found in the literature. Nevertheless, most of them do not completely exploit the CPU-GPU computing power as they usually tend to take advantage of thread and data parallelism available in GPU and only use one CPU thread to control the GPU activity [22].

Different approaches for GPU-based implementations of evolutionary algorithms are analysed in [13]. Some of them propose to implement all or the most of the steps of the evolutionary algorithm in GPU to decrease the cost of transferring information between CPU and GPU. In general, for an evolutionary algorithm, as the evaluation of fitness can be independently done for each individual in the population, this step is usually implemented in parallel. Nevertheless, other steps, such as the application of evolutionary operators, require interaction among individuals, thus involving some kind of synchronization among the computing elements. This way, two main researching lines can be distinguished among the proposals on a GPU implementation of evolutionary algorithms, i.e. a parallel implementation that shows the same behaviour than the sequential one, and the implementation of an evolutionary parallel algorithm tuned to the features of the GPU architecture. However, its characteristics could be different from those of the corresponding sequential algorithm. In this last alternative, an analysis of the suitability of the attained solutions should be done.

Paper [20] describes a CUDA implementation of a parallel genetic algorithm based on an island model. This paper is an example of the approaches that modify the evolutionary algorithm to reach a more suitable version for the available GPU architecture and resources. An alternative GPU implementation of the non-dominance rank used in NSGA-II, the Archived-based Stochastic Ranking Evolutionary Algorithm (ASREA), is provided in [21]. Paper [23] provides a parallel GPU implementation of a multi-objective evolutionary algorithm for a data mining application on marketing that predicts potential prospects from records of customers. This approach executes all steps of an NSGA-II algorithm in GPU except for the non-dominated selection, for which a fast procedure is proposed, and the non-dominated sort.

There are not many approaches using the CPU and GPU cores as resources that can be equally considered to distribute the workload of the optimization procedure. Paper [22] proposes a methodology to solve optimization problems in heterogeneous CPU-GPU architectures that benefit from both CPU and GPU cores, and points out the usefulness of further researching on this approach. Our procedure includes an evolutionary multi-objective optimization and a clustering algorithm applied to a set of high-dimensional patterns. Although the use of heterogeneous architectures including parallel data architectures such as GPUs has been proposed in previous papers, the parallelization on a heterogeneous platform of a whole data mining application with the characteristics of our target application is less frequent. Paper [10] analyses the effect of factors such as the communication patterns and the data partition on the performance of data mining applications, and in [3] a parallel multi-objective evolutionary procedure using MPI on only one platform is described. Our approach takes advantage of heterogeneous clusters including multiple CPU-GPU nodes and distributes the workload among both CPU and GPU cores of the nodes to speed up the application thus also allowing energy-saving.

## 5 EXPERIMENTAL RESULTS

In this section, we analyse the performance of our C++ codes, compiled with *mpic++* (GCC 4.8.5), running on Linux CentOS 7.4 operating system, in a cluster which contains four NUMA nodes connected by Gigabit Ethernet, each of them with 32 GB of DDR3 memory. One node (the front-end node), is dedicated to the master process, which is running on one CPU core, and also runs the sequential code. The characteristics of the OpenCL devices used in the other three nodes are as follows: Node 1 has two Intel Xeon E5-2620 v2 processors at 2.1 GHz (12/24 cores/threads) and a GPU Nvidia Tesla K20c with 5 GB of global memory, and 2,496 CUDA cores distributed into 13 SMXs. Nodes 2 and 3 are comprised of one GPU Nvidia Tesla K40m with 12 GB of global memory, and 2,880 CUDA cores distributed into 15 SMXs. On the other hand, nodes 2 and 3 contain, respectively, one and two Intel Xeon E5-2620 v4 processors at 2.1 GHz, thus comprising 8/16, and 16/32 cores/threads. Also, to perform the data parallelism of $K$-means algorithm in GPU, each SMX schedules 1024 work-items. In our experiments, we have used three datasets from the BCI Laboratory

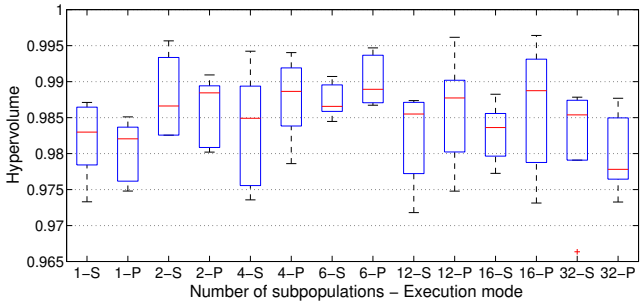


**Figure 2: (a) Hypervolume results of the (S)equential and (P)arallel execution modes**

at the University of Essex and described in [2]. They correspond to subjects coded as 104, 107 and 110, and each include 178 EEG patterns with 3,600 features per pattern. However, we only show the results for dataset 110 due to their similar results.

The implemented NSGA-II algorithm uses a uniform crossover with a probability of 0.75, mutation by inversion of the selected bit with a probability of 0.025, and selection by a binary tournament. The maximum value of the cost functions $f_1 = f_2 = 1.0$, and the hypervolume metric is calculated according to the Zitzler algorithm [24, 25], which uses (0,0) as the reference point, and thus, the maximum value is $hv = 1.0$. We have evaluated 3,840 individuals distributed into 1, 2, 4, 6, 12, 16, and 32 subpopulations, along 150 generations, including one local migration every 10 generations, and one global migration every 30 generations, and thus, a total of 5 global migrations and 15 local migrations.

The instantaneous power and energy consumption of the four nodes of our cluster have been measured by a watt-meter we have developed based on Arduino Mega. It provides, in real time, four measures per second for each node of our platform corresponding to the instantaneous power (in Watts) and the cumulated consumed energy (in $W \cdot h$) of the whole node. We do not measure the instantaneous power and energy consumed by the switch (in a future version of our watt-meter we will introduce a new sensor for this purpose) but we have checked that the instantaneous power is below 5 W. This way, it supposes a relatively low percentage of the instantaneous power and energy consumption in the nodes, and does not affect our conclusions.

Figure 2 provides the hypervolume obtained by the sequential and parallel modes for different subpopulations. It can be seen that the sequential and parallel modes provide hypervolumes quite near to the maximum value, 1.0, without significant differences among subpopulations and sequential/parallel modes. Moreover, it does not seem to be a trend in the variation of the means of the hypervolumes when the number of subpopulations changes. The values for 1 and 32 subpopulations in the parallel mode seem to be lower but the differences are not significant.

The improvement provided by the parallel mode with respect to runtime is shown in Figure 3. Figures 3.a and 3.b show the boxplots of runtimes for the sequential and parallel modes, respectively. In the sequential mode (Figure 3.a), although the changes can not be considered very significant, the runtime decreases when the number of subpopulations up to 4 and grows from 4 to 32 subpopulations. It has to be taking into account that in the sequential mode, one thread executes all code. As a subpopulation has fewer individuals is faster to compute a generation of the subpopulation, but there are more subpopulations (given a population with a fixed number of individuals), and the cooperation among these subpopulations also has to be processed. It seems that there is a trade-off among these two opposed trends for 4 subpopulations. In the parallel mode (Figure 3.b), except for 2 subpopulations, it is observed a decrease in runtime as more subpopulations are used. The differences are significant in all cases although between 1 and 4, 6 and 12 and 16 and 32 subpopulations are less important. The apparently anomalous behaviour observed for 2 subpopulations is related with the way that the subpopulations are assigned to the different nodes: as there are two OpenCL devices per node (see Figure 1), and the workload is distributed regarding subpopulations, only one of the nodes works when we have 2 subpopulations. The high decrease of runtime shown for more than 6 subpopulations can be explained by taking into account that from this number of subpopulations the three nodes of the platform are used (plus one CPU core of the front-end), giving a total of 37 CPU cores and 43 SMXs. In case of 4 subpopulations, only two nodes (besides the core of the front-end) work in the parallel code.

Figure 3.c shows the mean speedup achieved by the parallel alternative with respect to the corresponding sequential implementation. The speedup grows as more subpopulations are used except in case of 2 subpopulations whose behaviour can be understood from the runtime of Figures 3.a and 3.b. The maximum speedup is obtained for 32 subpopulations and is higher than 70 thanks to the benefit obtained from the achieved thread and data parallelism. Figures 4 and 5 correspond to energy consumption and instantaneous power of our sequential and parallel codes. Figures 4.a and 4.b give the energy consumed by, respectively, sequential and parallel codes. The shape of the boxplots shown in Figures 4.a and 4.b are similar to, respectively, those of Figures 3.a and 3.b (it has to take into account that the energy consumption depends on the product of power and execution time). The differences in energy consumption across subpopulations are quite similar to the sequential codes although a not quite significant reduction in energy consumption is shown for moderate numbers of subpopulations. The shape of energy consumption of parallel codes in Figure 4.b is also similar to the corresponding parallel runtimes shown in Figure 3.b.

Figure 4.c shows the rate of the mean energy consumed by the parallel code with respect to the corresponding sequential code. As can be seen, even in the worst case (2 subpopulations), the parallel code consumes only a 12% of the energy required by the sequential code, and thus, allows important energy-savings. We have obtained the best energy consumption in case of 32 subpopulations (less than 6% of the mean energy consumed by the sequential codes). From Figures 3.c and 4.c it is clear that parallel processing constitutes a valuable alternative not only to reduce runtime but also to decrease energy consumption, and thus opens interesting opportunities to improve the performance of evolutionary computation.

Figure 5 gives information about the evolution of the instantaneous power. Figure 5.a corresponds to the power consumption of the different nodes in the platform for the parallel code using 32 subpopulations. It is clear that the platform is heterogeneous because the maxima values of the curves change with the node.

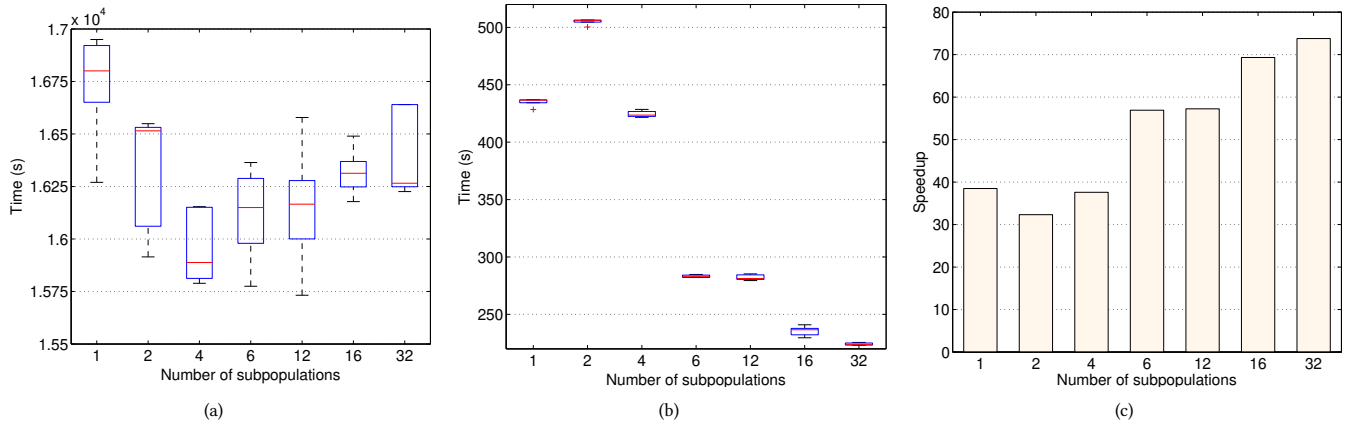


**Figure 3: Performance evaluation when increasing the number of subpopulations: (a) and (b) Running time of sequential and parallel modes, respectively; (c) Speedup achieved with respect to the sequential mode**

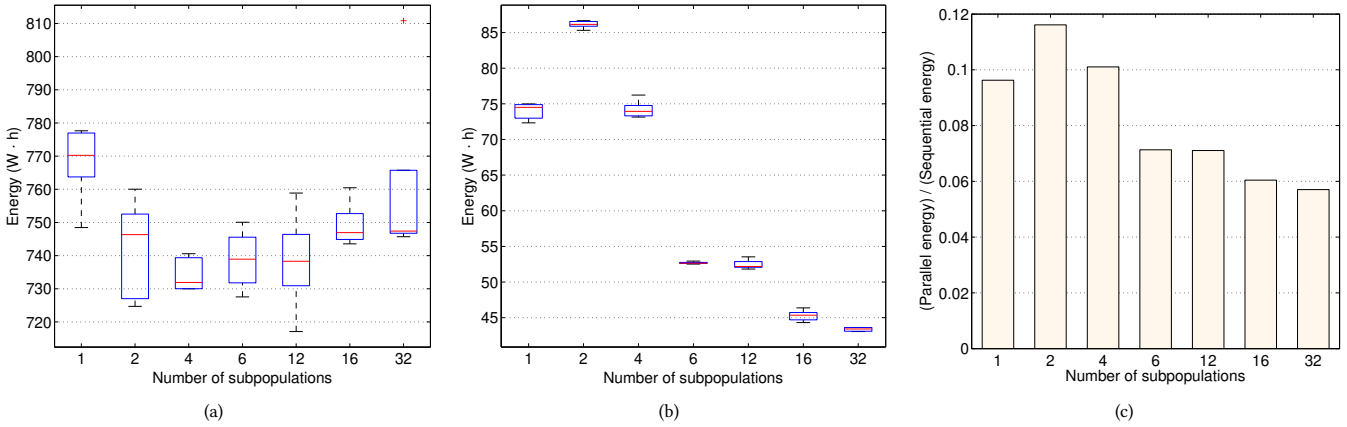


**Figure 4: Energy evaluation when increasing the number of subpopulations; (a) and (b) Energy consumed by sequential and parallel modes, respectively; (c) Energy rate required by the parallel mode with respect to the corresponding sequential mode**

Moreover, the changes in the power for different phases of the evolutionary algorithm are also apparent. For example, the falls in the instantaneous power of the 5 communications done among nodes. Figure 5.b provides the evolution of the instantaneous power of all nodes in the platform using 4, 16 and 32 subpopulations. From these curves is also apparent the fall in the instantaneous power due to the reduction in the activity of the cores in case of communication.

## 6 CONCLUSIONS

A procedure for multi-objective feature selection in EEG classification for BCI task has been parallelized to take advantage of heterogeneous clusters whose nodes include CPU and GPU cores. The corresponding code uses the MPI, OpenMP and OpenCL libraries to implement message-passing and shared-memory communication and benefit from thread-level and data-level parallelism across the cluster nodes, and their CPU and GPU cores.

The experimental results show that the proposed parallel approach for evolutionary multi-objective optimization is able not only to accelerate the execution runtime but also energy-saving with respect to a sequential implementation. Thus, speedups of more than 70 have been achieved requiring only about a 6% of energy consumed by the corresponding sequential code.

New studies could be useful to complete the experimental analysis of new alternatives and experimental situations. Although the instantaneous peak power for the switch is below 10% of the instantaneous power measured in the nodes, a detailed analysis of the energy consumption devoted to communications is also interesting. This way, an analysis about the possible trade-off of energy consumption and speedup according to the workload distribution among nodes and their cores could be done. Moreover, a more detailed study of the effect of the different steps of the parallel evolutionary algorithm in the instantaneous power would also be interesting to devise strategies for energy-efficient programming.

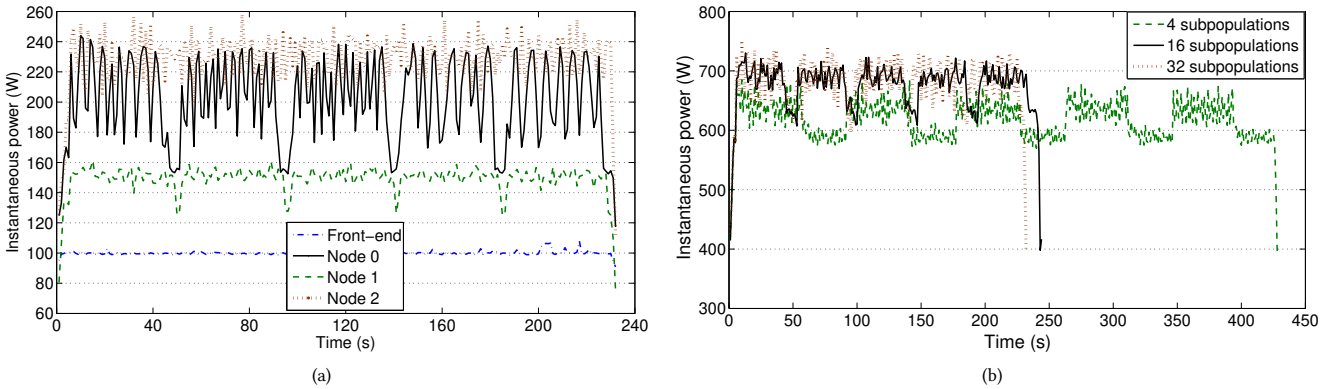


**Figure 5: Temporal evolution of the instantaneous power: (a) All nodes per separate, 32 subpopulations; (b) Total instantaneous power of all nodes, using a different number of subpopulations**

## ACKNOWLEDGMENTS

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.

## REFERENCES

[1] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J.M. Pérez, and I. Perona. 2013. An Extensive Comparative Study of Cluster Validity Indices. *Pattern Recognition* 46, 1 (2013), 243–256. https://doi.org/10.1016/j.patcog.2012.07.021

[2] J. Asensio-Cubero, J.Q. Gan, and R. Palaniappan. 2013. Multiresolution Analysis over Simple Graphs for Brain Computer Interfaces. *Journal of Neural Engineering* 10, 4 (2013), 21–26. https://doi.org/10.1088/1741-2560/10/4/046014

[3] C.A. Coello Coello and M. Sierra. 2004. A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. In *Proceedings of the 3rd Mexican International Conference on Artificial Intelligence (MICAI'2004)*. Springer, Mexico City, Mexico, 688–697. https://doi.org/10.1007/978-3-540-24694-7_71

[4] P. Collet. 2013. Why GPGPUs for Evolutionary Computation? In *Massively Parallel Evolutionary Computation on GPGPUs*, S. Tsutsui and P. Collet (Eds.). Springer, 3–14. https://doi.org/10.1007/978-3-642-37959-8_1

[5] J.J. Escobar, J. Ortega, A.F. Díaz, J. González, and M. Damas. 2017. Power-Performance Evaluation of Parallel Multi-objective EEG Feature Selection on CPU-GPU Platforms. In *Proceedings of the 17th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'2017)*. Springer, Helsinki, Finland, 580–590. https://doi.org/10.1007/978-3-319-65482-9_43

[6] J.J. Escobar, J. Ortega, J. González, and M. Damas. 2016. Assessing Parallel Heterogeneous Computer Architectures for Multiobjective Feature Selection on EEG Classification. In *Proceedings of the 4th International Conference on Bioinformatics and Biomedical Engineering (IWBBIO'2016)*, F. Ortuño and I. Rojas (Eds.). Springer, Granada, Spain, 277–289. https://doi.org/10.1007/978-3-319-31744-1_25

[7] J.J. Escobar, J. Ortega, J. González, and M. Damas. 2016. Improving Memory Accesses for Heterogeneous Parallel Multi-objective Feature Selection on EEG Classification. In *Proceedings of the 4th International Workshop on Parallelism in Bioinformatics (PBIO'2016)*. Springer, Grenoble, France, 372–383. https://doi.org/10.1007/978-3-319-58943-5_30

[8] J.J. Escobar, J. Ortega, J. González, M. Damas, and A.F. Díaz. 2017. Parallel high-dimensional multi-objective feature selection for EEG classification with dynamic workload balancing on CPU-GPU. *Cluster Computing* 20, 3 (2017), 1881–1897. https://doi.org/10.1007/s10586-017-0980-7

[9] F. Fernández-de-Vega, F. Chávez, J. Díaz, J.A. García, P.A. Castillo, J.J. Merelo, and C. Cotta. 2016. A Cross-Platform Assessment of Energy Consumption in Evolutionary Algorithms. In *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature (PPSN'2016)*. Springer, Edinburgh, UK, 548–557. https://doi.org/10.1007/978-3-319-45823-6_51

[10] A. Gainaru, E. Slusanschi, and S. Trausan-Matu. 2011. Mapping Data Mining Algorithms on a GPU Architecture: A Study. In *Proceedings of the 19th International Symposium. Foundations of Intelligent Systems (ISMIS'2011)*, M. Kryszkiewicz, H. Rybinski, A. Skowron, and Z-W. Raś (Eds.). Springer, Warsaw, Poland, 102–112. https://doi.org/10.1007/978-3-642-21916-0_12

[11] J. Handl and J. Knowles. 2006. Feature Subset Selection in Unsupervised Learning via Multiobjective Optimization. *International Journal of Computational Intelligence Research* 2, 3 (2006), 217–238.

[12] Khronos Group. 2015. Khronos OpenCL Registry. https://www.khronos.org/registry/cl/. (2015). Accessed: 2015-11-30.

[13] T.V. Luong, N. Melab, and E-G. Talbi. 2010. GPU-based Island Model for Evolutionary Algorithms. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'2010)*. ACM, Portland, OR, USA, 1089–1096. https://doi.org/10.1145/1830483.1830685

[14] S. Mittal and J.S Vetter. 2015. A Survey of CPU-GPU Heterogeneous Computing Techniques. *Comput. Surveys* 47, 4 (2015), 69:1–69:35. https://doi.org/10.1145/2788396

[15] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C.A. Coello Coello. 2014. A Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part I. *IEEE Transactions on Evolutionary Computation* 18, 1 (2014), 4–19. https://doi.org/10.1109/TEVC.2013.2290086

[16] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C.A. Coello Coello. 2014. A Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part II. *IEEE Transactions on Evolutionary Computation* 18, 1 (2014), 20–35. https://doi.org/10.1109/TEVC.2013.2290082

[17] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou. 2017. A Survey of Power and Energy Predictive Models in HPC Systems and Applications. *Comput. Surveys* 50, 3 (2017), 37:1–37:38. https://doi.org/10.1145/3078811

[18] OpenMP Community. Accessed: 2016-11-21. OpenMP specifications. http://www.openmp.org/specifications/. (Accessed: 2016-11-21).

[19] J. Ortega, J. Asensio-Cubero, J.Q. Gan, and A. Ortiz. 2016. Classification of motor imagery tasks for BCI with multiresolution analysis and multiobjective feature selection. *BioMedical Engineering OnLine* 15, 1 (2016), 73. https://doi.org/10.1186/s12938-016-0178-x

[20] P. Pospichal, J. Jaros, and J. Schwarz. 2010. Parallel Genetic Algorithm on the CUDA Architecture. In *Proceedings of the 13th European Conference on the Applications of Evolutionary Computation (EvoApplications'2010)*, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A.I. Esparcia-Alcazar, C-K. Goh, J.J. Merelo, F. Neri, M. PreuSS, J. Togelius, and G.N. Yannakakis (Eds.). Springer, Istambul, Turkey, 442–451. https://doi.org/10.1007/978-3-642-12239-2_46

[21] D. Sharma and P. Collet. 2013. Implementation Techniques for Massively Parallel Multi-objective Optimization. In *Massively Parallel Evolutionary Computation on GPGPUs*, S. Tsutsui and P. Collet (Eds.). Springer, 267–286. https://doi.org/10.1007/978-3-642-37959-8_13

[22] P. Vidal, E. Alba, and F. Luna. 2017. Solving Optimization Problems Using a Hybrid Systolic Search on GPU Plus CPU. *Soft Computing* 21, 12 (2017), 3227–3245. https://doi.org/10.1007/s00500-015-2005-x

[23] M.L. Wong and G. Cui. 2013. Data Mining Using Parallel Multi-objective Evolutionary Algorithms on Graphics Processing Units. In *Massively Parallel Evolutionary Computation on GPGPUs*, S. Tsutsui and P. Collet (Eds.). Springer, 287–307. https://doi.org/10.1007/978-3-642-37959-8_14

[24] E. Zitzler. 1999. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Shaker Verlag Germany.

[25] E. Zitzler and L. Thiele. 1998. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*. Springer, Amsterdam, The Netherlands, 292–301. https://doi.org/10.1007/BFb0056872