The Journal of Supercomputing (2019) 75:6194–6219
https://doi.org/10.1007/s11227-019-02843-4

A component-based study of energy consumption for sequential and parallel genetic algorithms

Amr Abdelhafez[1] · Enrique Alba[1] · Gabriel Luque[1]

Published online: 4 April 2019

Abstract
Recently, energy efficiency has gained attention from researchers interested in optimizing computing resources. Solving real-world problems using optimization techniques (such as metaheuristics) requires a large number of computing resources and time, consuming an enormous amount of energy. However, only a few and limited research efforts in studying the energy consumption of metaheuristics can be found in the existing literature. In particular, genetic algorithms (GAs) are being used so widely to solve a large range of problems in scientific and real-world problems, but hardly found explained in their internal consumption behavior. In the present article, we analyze the energy consumption behavior of such techniques to offer a useful set of findings to researchers in the mentioned domains. We expand our study to include several algorithms and different problems and target the components of the algorithms so that the results are still more appealing for researchers in arbitrary domains of application. Our experiments on the sequential GAs show the controlling role of the fitness operator on energy consumption and also reveal possible energy hot spots in GAs operations, such as mutation operator. Further, our distributed evaluations besides a statistical analysis of the results demonstrate that the communication scheme could highly affect the energy consumption of the parallel evaluations of the GAs.

Keywords Energy consumption · Green computing · Genetic algorithms · Sequential · Parallel

✉ Amr Abdelhafez
amr@lcc.uma.es

Enrique Alba
eat@lcc.uma.es

Gabriel Luque
gabriel@lcc.uma.es

[1] Dpto. de Lenguajes y Ciencias de la Computación, Univ. de Málaga, E.T.S. Ingeniería Informática, Campus de Teatinos, 29071 Málaga, Spain

1 Introduction

In today's world, energy efficiency is an important topic in all scientific areas. In computer science, a new domain called *green computing* [19] has emerged to deal with the efficient use of computing resources to reduce the environmental footprint. The electricity consumption of computing devices has a large impact in our current digital era, and it will increase in the next years. Although electricity is an environment-friendly form of energy, its generation and distribution emit a significant amount of pollutants [e.g., carbon dioxide (CO_2) and sulfur dioxide (SO_2)]. These pollutants increase global warming and represent a threat to any living being [26, 27].

Building real green computing applications requires the knowledge of the actual consumption of each component of their algorithms or software pieces [1]. However, measuring the energy consumption of software still faces many practical and theoretical problems, e.g., the lack of specialized software able to accurately measure energy consumption (hardware independent) [18]. Even the actual building of software that uses this information is quite unprecise [36]. Among the most consuming type of programs we can find out there, search techniques when solving complex problems have a prominent place. Search techniques require large computation times and are frequently run to optimize daily activities in cities and factories [14]. Genetic algorithms (GAs) are search, optimization, and learning methods that have been widely applied in many research areas and real-life problems, e.g., engineering, aircraft design, optimization, logistics, bioinformatics, scheduling, robotics.

The availability of high-performance computing (HPC) resources makes GAs suitable for parallelism in order to solve more complex and lager problems [4]. In the literature, there are no studies that offer precise indications on where is energy going in GAs and parallel GAs (PGAs), what would allow the next phase of smart use of this information to build efficient algorithms with lower consumption for a similar numerical result.

In this work, we focus on the measurement and quantitative analysis of the energy consumption of GAs and PGAs. We perform an extensive analysis of the energy consumption and execution time of the components of a sequential (panmictic) genetic algorithm. Further, we include the energy consumption analysis of PGAs (a distributed GA (dGA) in our case) with two different communication schemes. We aim to answer four significant research questions:

RQ1 Which GA component consumes the most?
RQ2 What is the effect of the problem size on the energy consumption of GA components?
RQ3 What is the effect of using different communication schemes on the energy consumption of the dGA?
RQ4 What is the impact of using a variable number of cores on the global energy consumption of the dGA?

For RQ1, we analyze the energy consumption of GA components one by one using a varied number of problems and dimensions. For RQ2, we test problems with different dimensions to study the effect of the size of the problem on the energy consumption of different GA components. For RQ3, we extend this analysis to a dGA with two different communication schemes (synchronous and asynchronous). RQ4 requires an extensive study to the energy consumption features of the dGA over a different number of cores from 1 to 32. We consider a varied set of problem features (dimensionality, search landscape, multimodality, etc.) so that meaningful conclusions are feasible.

In short, the main contributions of this paper are to study the energy consumption and execution time behavior of both sequential and parallel GAs, in reasonably wide energy analysis of its components under different computer communication schemes. We expect that our research (energy understanding of GAs) will be a spot of interest to the research community. The potential impact of these results in building new techniques that fit the aims of green computing will also link to a line of research for making algorithms more efficient as a piece of software running on a computer. This point of view is not so present in the literature, but a fundamental one for formulating high-quality research.

The rest of this paper is organized as follows: Sect. 2 discusses several related works for an overview of their recent advances. Section 3 defines the basic concepts of the canonical and distributed GA, with a description of the different dGA communication schemes. Section 4 provides the methodology for our experiments with a justification of the problems used in the experiments. In Sect. 5, we present our numerical results to analyze the sequential and distributed GA with a discussion on it. Section 6 summarizes the conclusions and discusses open research lines.

2 Related works

Overall, the literature on analyzing energy consumption of GAs exists, though it is very limited yet. As to PGAs, the literature is still shorter. We can also find several articles on nearby fields, like applying metaheuristics for problems dealing with energy minimization in a target scenario. However, that approach is different from our goal since they are not targeting the energy consumption of the algorithm itself but using the algorithm to solve problems concerns energy.

In the area of GAs, there are efforts to study the energy consumption behavior of the algorithm. One of these efforts was proposed by [13]. There, authors developed a GA to solve an extended version of the job-shop scheduling problem by considering its energy consumption. Another related effort was performed in [16], where authors evaluated the performance of three metaheuristic algorithms on the basis of cost and minimizing energy reductions. In the area of parallel algorithms, the authors of [25] proposed a parallel bi-objective hybrid genetic algorithm that takes into account energy consumption. They studied island and multi-start parallel GA models with a hybrid approach between a multi-objective PGA and energy-conscious scheduling heuristic. They concluded that the hybrid approach consumes more resources than the energy-aware scheduling heuristic, and the insular approach consumes more resources than the hybrid approach.

Previously mentioned efforts studied GAs and other algorithms to improve energy efficiency when solving a problem. However, they did not analyze the energy consumed by the GA or its components. In the scope of analysis of energy consumption of evolutionary algorithms (EAs), Vega et al. [38] presented a preliminary study on the energy consumption of the genetic programming (GP) algorithm. They run their experiments on different hardware devices over a number of operating systems. The main goal of their study was to show the effect of the main parameters of the algorithm on the energy consumption. They concluded that devices with better processors can run the algorithm faster but spend larger amounts of energy. They also reported the influence of changing population sizes in the variable amount of energy required to reach solutions. Another recent research in this regard was proposed by Alvarez et al. [7]. The authors of that work presented a preliminary energy consumption estimation model, based on the analysis of the influence of GP parameters on their energy consumption under a number of hardware devices. They concluded that their model was able to correctly estimate the energy consumption of the GP algorithm over different devices.

We now turn to review the relevant researches concerning tools to measure energy consumption, either in algorithms or in other types of software. Since it was proposed by Intel, Intel's running average power limit (RAPL) [10] interface has been used widely to measure the energy consumption of algorithms. In [9], authors used RAPL as a measurement to the execution of the algorithms and stated that it is reliable in many different types of computing systems. In another aspect, the authors of [39] used the RAPL interface to measure the energy consumption characteristics of MPI calls. They proposed a model to accurately measure the aggregate energy consumed by all processes engaged in MPI operations. In [31], authors extended Flex-MPI [24] with energy-aware and power-aware capabilities. Their aim was to increase the energy efficiency of parallel applications by means of malleability. All the power measurements in their novel approach were obtained by means of the RAPL interface. With a different objective, the work presented in [30] focused on the comparison between RAPL and two other power-meter methods for gathering energy consumption values. Their contribution was to study the correspondence or difference of the energy data provided by these methods. Their analyses show that RAPL has good correspondence due to using the faster and reliable hardware counters, which allows to use it for measuring energy consumption accurately. For more researches on using RAPL in energy measurement, we refer the reader to [8, 21, 28]. Thus, we could claim that RAPL proved to be a reliable tool for measuring the energy of algorithms.

In summary, there are several differences between the existing works and our approach. We here present a study on the energy consumption behavior of sequential and distributed GAs, not GP or other ad hoc heuristics or software. Also, the previous works were dealing with either an application where energy was considered (out of the algorithm itself) or tools for measuring energy in software packages in general. The actual situation in this domain is that of a definite shortage of works focusing on GAs (a really important kind of techniques today) with modern tools for measurement. Besides, GA and PGAs are structurally different in many aspects, so they need and deserve a focused study, as we do here.

3 Genetic algorithms background

In this section, we provide the background algorithmic information needed to understand and reproduce this work. In particular, we present an overview of the canonical panmictic GA and dGA.

3.1 Overview of the canonical genetic algorithm

GA starts with a randomly generated set of individuals, called a *population*. Each *individual* (chromosome plus fitness) represents a possible tentative solution. Each *chromosome* is composed of an array of genes depending on the dimension of the problem solved. The *fitness* (optimized objective) function is used to evaluate the quality of every individual in relation to the rest. Genetic *operators* (usually selection, crossover, mutation, and replacement) are used to generate new solutions for the next generation. This process is performed until the stopping criterion is met (maximum number of fitness evaluations or find a solution of good quality). Algorithm 1 provides the pseudocode of this panmictic algorithm.

Algorithm 1 The Canonical Genetic Algorithm

```
1: Initialization. Generate randomly an initial population $P$.
2: Evaluation. Evaluate the individuals in $P$.
3: while $not\ stop - condition$ do
4:     P′ := Variation operators [P];
5:     Evaluate fitness [P′];
6:     P″ := Select [P′];
7: end while
```

The term *panmictic* means that all the individuals in the same single population can probably mate to the rest, i.e., there is no restriction to their interactions. In the *structured* GAs [5, 34], the individuals are geographically separated, and interactions occur only inside these isolated neighborhoods. Multi-population GAs (such as dGA) are the typical example of structured GAs. dGA can indeed be run in parallel on different cores (or not), but this refers to its physical execution, not to the design of the algorithm (as *distributed* points out).

3.2 The distributed genetic algorithm

In our study, we also consider the distributed model largely described in [4] and many other works. In this model, the population is divided into many subpopulations called *islands* all having the same number of individuals. These islands can run in parallel (so we could call it PGA or parallel dGA) and can exchange information and knowledge among them. The resulting behavior of this parallel dGA is quite different from the canonical one and in general beneficial to explore different parts of the search space at the same time (distributed effect) plus a significant reduction in the time expected from performing more operations per time unit (parallel effect). Once migration conditions are met, individuals (or other information) occasionally migrate from one particular island to its neighbors. (So, a topology is needed.) In Algorithm 2, we provide the pseudocode of a canonical dGA.

Algorithm 2 The Distributed Genetic Algorithm

```
1: Island $i$, $t$:=0;
2: **Initialization.** Generate randomly an initial population $P_t$.
3: **Evaluation.** Evaluate the individuals in $P_t$.
4: **while** $not\ stop - condition$ **do**
5:     P′(t) := Variation operators [P($t$)];
6:     Evaluate fitness [P′($t$)];
7:     P′′ ($t$) := Select [P′($t$)];
8:     P′′′($t$) := Communications between islands [$\Delta$i, P′′($t$)];
9:     $t := t + 1$;
10: **end while**
```

It is clear that individuals only mate inside islands, and the sparse migrations make it possible for a copy of them to arrive at other islands and merge then with locals. The time between successive communications, Δi, is one of the most important parameters of the migration policy, regulating the degree of connectivity and indeed the potential consumed energy for computing and using the communication network.

Our migration topology is a unidirectional ring topology, a very common one. (We want to maximize our impact in the field by using common models in this first paper on energy.) Thus, an island can send and receive migrants only from its next and previous neighbors, respectively. The actual communications between islands do not only depend on frequency or topology, but it could also vary according to the synchronism of the implementation. The most common dGA communication policies are synchronous and asynchronous schemes [3].

3.3 Synchronous and asynchronous communication schemes

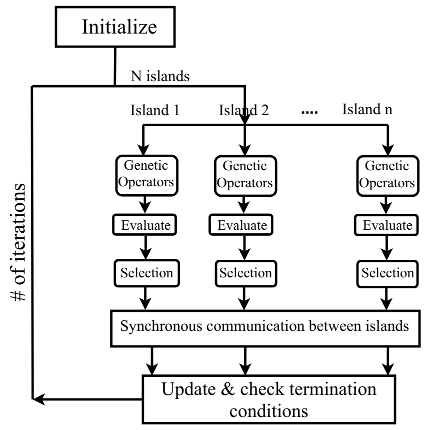
In our experiments, we will address both synchronous and asynchronous implementations for the same case studies. Studying both implementations present an impact in the field than previous studies since most existing works usually address one implementation. Synchronous versions are common when the algorithm is run on shared memory for all islands since all islands should then proceed at a similar pace. Asynchronous distributed genetic algorithms (dGAs) are very common in Internet systems policies and clusters with heterogeneous computing units.

Synchronous and asynchronous implementations are governed by the same algorithm and parameters. The main difference is in the communication scheme: Either we proceed at the same pace (sync) and wait for the slower island (in every generation) of the dGA or every island advances at its own pace (async) and incorporates incoming information as soon as it arrives. Figure 1 shows the differences between the two schemes in a graphical manner.

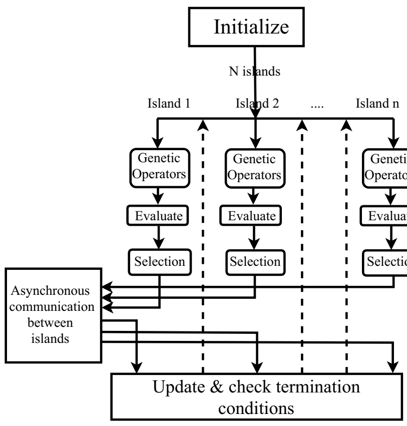
The main difference between these implementations is that the synchronous version has a synchronization point at every migration interval, where all the islands exchange the search information simultaneously [3]. At that point, all the processes should wait and block for other processes to reach this point. So, it is highly probable that at every communication point there are some idle processes waiting for some others to finish their task. It is important to mention that the migrant solutions are from the same evaluation phases, which may lead to lower performance for the migration operator.

On the contrary, in the asynchronous approach there are not such synchronization points; thus, islands proceed on their own with sparse and non-synchronous exchanges of information once arrived at the island. Every island includes the received individuals from migration whenever possible, then avoiding any waiting [3]. Thus, this communication scheme potentially evolves immigrant solutions independently from the different generation times among the islands. This way of sharing knowledge gives more diversity to the islands across the search process, promoted by the different speeds of the computing units; thus, there are no idle processes by construction.

Even if the synchronized algorithms might have some performance issues, they are spread. Synchronization of communications may cause some resources to be idle waiting for communications [2], but in some problems (due to a specific problem requirement) it may come as a design choice. Therefore, studying the



(a) Synchronous scheme



(b) Asynchronous scheme

Fig. 1 Functional diagram for the synchronous/asynchronous dGA schemes

effect of synchronism on the parallel and distributed algorithms is an important issue, never previously done in connection with their energy consumption.

4 Design of experiments

Our approach is to present energy consumption and execution time studies of the (panmictic) sequential GA and the (structured) parallel dGA. As to the sequential GA, we study the energy consumption and execution time of the GA components (not just of the whole algorithm). In the second analysis, we expand the study to include energy consumption of the dGA with different communication schemes. Both studies provide knowledge to researchers not just on the actual internal behavior in terms of energy, but they represent an opportunity to build less consuming algorithms in future studies.

Let us start with the sequential GA. Here, we will justify why and how we measure the energy consumption of the following components:

- **Fitness evaluation:** This operator is one of the main GA components since it is used for calculating the quality of solutions and guiding the search. A fitness value indicates how close a solution to the optimal solution is. The computational resources required by this operation highly depend on the problem being solved. Other problem characteristics, e.g., multi-objective optimization and dynamic problems, have a strong relation to fitness computation; not to mention parallelism, that comes handy in many cases because of its high computational demands. It is now clear the importance of the study for energy consumption of evaluations apart.
- **Genetic operators:** Usually, the variation operators of a GA are used to generate new solutions based on the existing ones. The active variation operators in a GA are crossover and mutation. The aim of these operations is to modify the current population to get a new one. The crossover (or recombination) operator combines two or more different solutions to generate new solutions, while the mutation operator modifies the solution by changing its genes to generate a new one. The main role of the mutation operator is to add genetic diversity inside the population, thus preventing the algorithm from converging to a local optimum. Genetic operators are substantial to GAs; they deserve an energy profiling study.
- **Housekeeping:** This term (in our study) refers to all the rest of the tasks of the algorithm, e.g., initialization, selection, and I/O operations. These operations have fewer computations and instructions to be executed compared to the previous operators.

Further, we present an analysis of the energy and time consumption of the synchronous and asynchronous dGAs over a different number of cores. We use a set of problems with different features and sizes. Moreover, we perform a statistical comparison for the energy consumption behavior between these two implementations. Our algorithms have been implemented in C++, and the communication phase in the dGA is implemented using the MPI message-passing programming model. The communication topology uses MPI's blocking/non-blocking communication function features for the synchronous and asynchronous migrations, respectively. MPI allows a natural and easy partitioning of the problem, and it provides portability and efficiency.

All the energy measurements in our experiments were obtained by means of RAPL [10]. The RAPL interface is implemented in C++ and invoked from the source code. The reasons for using RAPL are many: First, it has already been proven to be stable and to achieve high accuracy energy estimates at a fine-grained level [17, 32, 41]. Second, it does not require built-in hardware on Intel-compatible systems. Third, RAPL measures the energy consumption of the running code without introducing influent overhead.

4.1 Benchmark problems

In order to perform a comprehensive energy consumption study, we use a set of eight different problems with many instances each. These problems vary in dimension sizes, computational complexity, and search landscapes, which will lead to better-grounded conclusions. The problems used in our experiment are as follows: ONEMAX problem [33] (or bit-counting), multimodal problem generator (P-PEAKS) [20], error-correcting code design (ECC) [23], the minimum tardy task problem (MTTP) [35], COUNTSAT [12] which is an instance of MAXSAT, maximum cut of a graph (MAXCUT) [22], frequency modulation sounds (FMS) [37], and the massively multimodal deceptive problem (MMDP) [15]. They represent quite a diverse set, not only from a numerical point of view but also showing different complexities. Table 1 gives a list of the problems sorted according to the evaluation operator complexity.

ONEMAX (bit-counting problem) is a basic test function used for evaluating and comparing algorithms; it will allow these results to benefit other papers in theory of GAs and will provide a basic line for comparison. The multimodal problem generator P-PEAKS employs n-bit solutions named *peaks*. These solutions represent the location of the *peaks* in the search space. The fitness of a given tentative solution is computed using the Hamming distance to the nearest peak in the search space [11]. ECC is an important problem in the area of secure communications (error code design); our instance consists of 24 words and a word length of 12 bits. MAXCUT, COUNTSAT, and MTTP are NP-complete combinatorial problems of interest in the field of metaheuristics used for testing and evaluating algorithms. FMS is a real-world problem in the field of engineering. FMS has six real-valued parameters of a

Table 1 Benchmark problems and its evaluation operator complexity

Problem	Evaluation	Problem	Evaluation
COUNTSAT	$O(n)$	MAXCUT	$O(n^2)$
MTTP	$O(n)$	P-PEAKS	$O(n^2)$
FMS	$O(n)$	ECC	$O(n^2)$
ONEMAX	$O(n)$	MMDP	$O(n^2)$

frequency-modulated sound model, and the goal is to minimize the sum of squared errors from the proposed solution to a reference model. The MMDP problem is composed of k deceptive subproblems, with the goal of cheating the search algorithm and making it perform badly. We use one instance with $k = 60$ subproblems.

In all of our benchmark problems, we are maximizing the objective function, except for FMS where we are minimizing it. We here need to remind that our goal in this paper is not to offer the best performing algorithm for these problems, but to use problems from the literature that will allow our energy consumption to be more representative than usually found till now. This approach is a diverse source of search features that would require from the algorithm an unknown and very different performance.

4.2 Parameter settings and system specifications

In this section, we present the parameters used in our experiments. We determined these values by a set of preliminary numerical experiments, with the goal of allowing the running to expose different behaviors and energy profiles, for richer analysis. Table 2 presents the values used for the sequential GA and parallel dGA.

Our results are the average of 30 independent runs of the algorithms, executed in a dedicated multicore computer. These multiple runs allow also reducing the effect of system and I/O overhead in the measurements. We conducted our experiments on an Intel-based PowerEdge T430 Tower Server with the following specifications: Intel(R) Xeon(R) E5-2620 v4 2.10 GHz, 20M Cache, Linux Ubuntu Server 14.04.5 LTS operating system, and 64GB of RAM.

5 Numerical experiments

This section presents the results of the introduced algorithms to solve the explained problems. First, we analyze the results of the energy consumption of the sequential GA components, showing how much each component consumes. Later, we present

Table 2 Parameter settings for the sequential and parallel dGA

Definitions	Values
Population size in the sequential GA	100 individuals
Subpopulation size in the dGA	50 individuals
# of islands in the dGA	32 islands
Crossover	Uniform, $pc = 0.6$
Mutation	Bit-flip, $pm = 0.1/n$
Selection operator	Binary tournament
Replacement	Replacing the worst
Elitism	Yes
Migration rate	2 individuals
Migration gap	5000 evaluations

Table 3 Mean of the energy consumption of GA components for 30 independent runs, in kWh

Problem	Dimension	Evaluation	Crossover	Mutation	Housekeeping	Total
COUNTSAT	20	<u>6.54E−04</u>	<u>7.87E−04</u>	7.97E−04	<u>7.30E−04</u>	2.97E−03
MTTP	100	1.15E−03	2.05E−03	2.54E−03	8.09E−04	6.55E−03
FMS	288	2.22E−02	5.37E−03	7.45E−03	1.13E−03	3.61E−02
ONEMAX	2000	3.95E−03	**3.70E−02**	**5.15E−02**	9.63E−03	1.02E−01
MAXCUT	100	**3.05E−02**	1.99E−03	2.60E−03	<u>7.95E−04</u>	3.59E−02
P-PEAKS	100	**3.06E−02**	2.00E−03	2.73E−03	<u>5.34E−05</u>	3.54E−02
ECC	288	**3.25E−02**	5.38E−03	7.54E−03	1.07E−03	4.65E−02
MMDP	360	1.56E−03	6.65E−03	9.30E−03	1.42E−03	1.89E−02

Table 4 Energy consumption percentages (%) of the different GA components

Problem	Evaluation	Crossover	Mutation	Housekeeping
COUNTSAT	22.02	26.52	**26.87**	24.59
MTTP	17.62	31.30	**38.74**	12.34
FMS	**61.38**	14.86	20.64	3.12
ONEMAX	3.87	36.25	**50.45**	9.43
MAXCUT	**85.01**	5.53	7.25	2.22
P-PEAKS	**86.48**	5.66	7.71	0.15
ECC	**69.90**	11.58	16.22	2.31
MMDP	8.22	35.13	**49.15**	7.51

Boldfaced values represent the component with the highest energy consumption by problem

an analysis of the synchronous and asynchronous parallel dGA to spot its unique energy profiles.

5.1 Energy consumption analysis of the sequential genetic algorithm

In this analysis, we focus on the energy consumption and execution time analysis of GA components. We aim to present a detailed answer to our first research question (RQ1) proposed in the Sect. 1. Tables 3 and 4 present the energy consumption in kWh, and energy consumption percentages of the GA components, respectively. We separate problems of $O(n)$ and $O(n^2)$ in the table with a dashed line. These results are the average of the 30 independent runs of the algorithm after 10^6 generations. In Table 3, we show in boldface the five highest energy consumption components and underline the lowest energy consumption components among all the problems. In Table 4, we show in boldface the components with the highest energy consumption percentages for each problem, respectively.

As to the total energy consumption values reported in Table 3, the first conclusion is that values vary depending on the problem dimension and fitness function complexity. For MAXCUT and P-PEAKS problems, the total energy consumption of the algorithm is similar, since the two problems have the same dimension size and evaluation operator complexity. For problems COUNTSAT and ONEMAX, the GA scores the lowest and highest energy consumption amount, respectively. This happens again because of the dimension size of each problem. For the evaluation operator, ECC scores the highest energy consumption amount among all the evaluation values out of all the other problems. The reason for this is the complexity and implementation of the ECC problem, which requires more computations in the evaluation process. The controlling role of the dimension size is very clear in the genetic operators. For problems of dimension 100 (P-PEAKS, MTTP, and MAXCUT), genetic operators score approximately the same amount of energy; the same behavior is detected in the case of dimension 288 (ECC and FMS). The reason for this behavior is that these operators are applied to the population regardless of the fitness operator complexity.

In Table 3, we can observe that three of the highest five energy consumption values (marked in bold) are for the evaluation operator of MAXCUT, P-PEAKS, and ECC problems. These problems have a complexity of $O(n^2)$, which requires more computation and time in the evaluation operator. MMDP has also a quadratic complexity, but its evaluation can be implemented in a very efficient way, reducing the amount of energy consumed for the studied instances. The other two values are for the crossover and mutation operators of ONEMAX problem. Since all the operators have linear complexity, the dimension is the most important factor in the energy consumed by them. As we remarked above, ONEMAX has the highest dimension among all the problems, which requires higher energy consumption in the genetic phase. We can also observe that three of the least five energy consumption components (marked with underline) are for the COUNTSAT problem, again appear the dimension of the problem as a leading role for the energy consumption value.

In Table 4, we also show in boldfaced the components with the highest energy consumption percentages for each problem. In the case of percentages, we can distinguish two different behaviors: The evaluation operator is the one which consumes the most energy for four problems, while in the other four problems, the most energy consumption operators are the mutation. These results answer our RQ1 and besides confirm the leading role of the evaluation operator. It also confirms another important observation that mutation represents a potential energy hot spot component inside the GA. The consumption of the evaluation operator is clear due to its higher complexity with respect to the rest of the components. The mutation consumption is, however, more subtle. This operator is constantly generating random numbers to decide whether a bit should be flipped or not. In our experiments, we employ the standard random number generation function provided by the programming language which is the most common approach in the GA implementations. The random number generation is shown to be quite an expensive operator (in time and energy) and the reason for the mutation consumption behavior. Housekeeping represents the smaller energy percentages in seven out of our eight problems: less than 15% of the total energy consumed by the algorithm. The only exception is COUNTSAT, where it goes up to 24.59% of the total energy consumption. This is because the component has some operations which are independent of the features of the problem, and for easy problems (low dimensionality) with low energy consumption, this fix cost

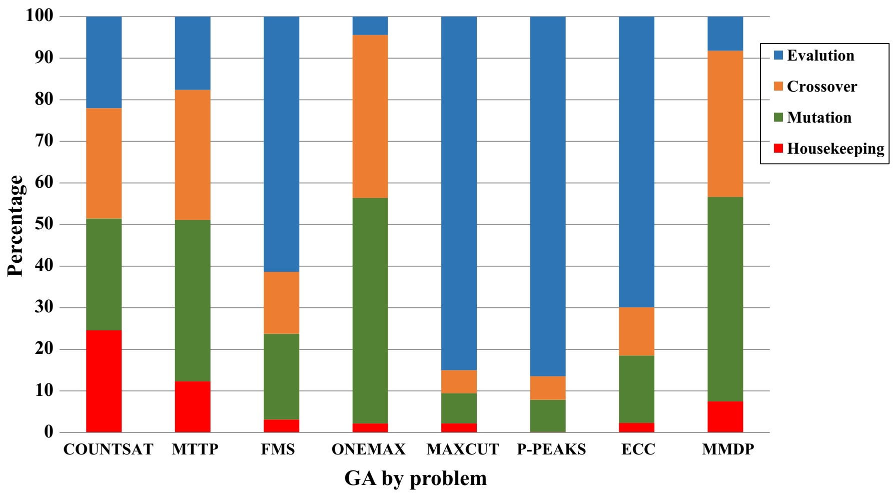


Fig. 2 Energy consumption percentages (%) of the different GA components

Table 5 The execution time percentages (%) of the different GA components

Problem	Evaluation	Crossover	Mutation	Housekeeping
COUNTSAT	24.30	25.30	**25.40**	25.00
MTTP	18.28	31.06	**37.82**	12.84
FMS	**62.22**	14.56	20.30	2.92
ONEMAX	4.21	35.26	**50.56**	9.98
MAXCUT	**85.85**	5.17	6.78	2.21
P-PEAKS	**87.36**	5.25	7.21	0.18
ECC	**72.42**	10.62	14.98	1.99
MMDP	8.63	35.07	**49.19**	7.11

could be important when it is compared with the cost of the classical GA operations. These results are intuitively shown in a graphical form in Fig. 2, and we can see how much each component consumes compared to the rest in the algorithm. These percentages show a variety of energy consumption values for the different problems, as it was expected.

We conclude this analysis by showing the execution time percentages of each component in Table 5. The results shown in this table are clearly similar to the energy consumption percentages shown in Table 4. Also, Figs. 2 and 3 resemble each other for most of the problems. The correlation between execution time percentages and energy consumption percentages is a common sense finding, since energy (E) is directly proportional to time (T) and power (P) ($E \approx P \times T$). We should take into account that P is almost in the same range of watts but not a constant, so these percentages will not be exactly the same in both cases of energy and time percentages.

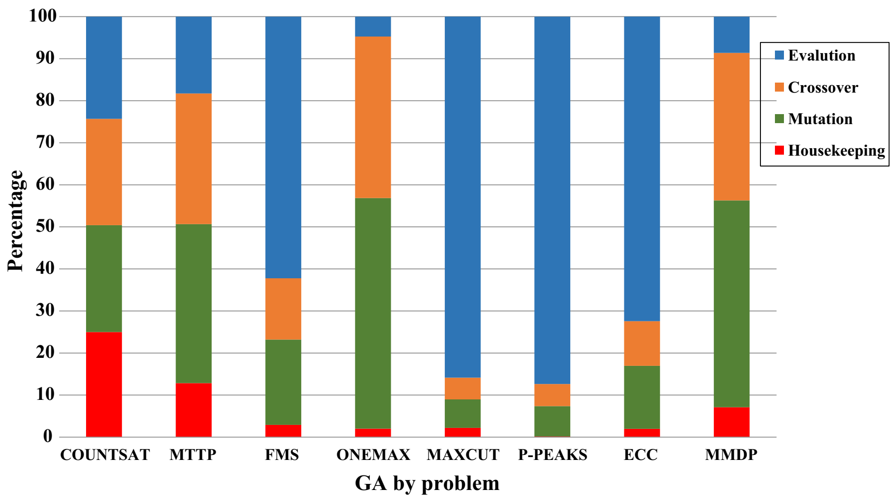


Fig. 3 Execution time percentages (%) of the different GA components

The discussion of the results presented in this section fully explains the answers to the first research question (RQ1). It shows the key influence of the evaluation in the complex problems. Also, it hints on the importance of studying other operations (mainly for classical mutation operators) for developing an energy-aware algorithm.

5.2 Dimensionality and energy consumption

In this section, we will answer and clarify RQ2, discussing the effect of the problem size on the energy consumption of the GA components. There are many factors that could affect the scalability analysis of energy consumption, e.g., the parameters (especially population size) and the search space model. In consequence, we have selected two different problems (ONEMAX and P-PEAKS) varying in search space model and fitness function complexity.

ONEMAX is well known and commonly used as a benchmark problem with linear fitness function complexity, which ensures a better understanding of the direct effect of the problem size on energy and time consumption. P-PEAKS (a multimodal problem of $O(n^2)$) has a tunable range of epistasis, thus allowing to generate instances with growing difficulty when the problem size changes [11]. We evaluate our algorithms on seven instances ranging from ten till 3000 bits, to allow different behaviors of the instances emerge and thus present a robust conclusion. We have fixed other GA parameters (Table 2) for all instances while changing only the problem size. We must highlight that P-PEAKS has two main parameters: the number of bits of each peak and the number of peaks. In this experiment, we change the number of bits, while the number of peaks remains fixed (100). Our results are the average of 30 independent runs of the algorithm for 10^6 generations.

Tables 6 and 7 present a scalability analysis to the increment of the dimension on the energy consumption and execution time for ONEMAX problem. The boldfaced

Table 6 Energy consumption percentages (%) of the GA components for ONEMAX problem

Dim	Evaluation	Crossover	Mutation	Housekeeping
10	24.25	25.54	**26.09**	<u>24.11</u>
100	<u>11.68</u>	32.64	**43.01**	12.67
200	<u>7.15</u>	35.72	**48.64**	8.50
500	<u>4.51</u>	36.91	**50.80**	7.78
1000	<u>4.41</u>	39.16	**54.26**	2.17
2000	<u>3.87</u>	36.25	**50.45**	9.43
3000	<u>3.80</u>	34.84	**48.15**	13.20

Table 7 Execution time percentages (%) of the GA components for ONEMAX problem

Dim	Evaluation	Crossover	Mutation	Housekeeping
10	26.85	26.79	**27.43**	<u>18.93</u>
100	13.09	31.76	**42.10**	<u>13.05</u>
200	<u>7.89</u>	35.10	**48.79**	8.23
500	<u>4.89</u>	36.39	**51.45**	7.27
1000	<u>4.73</u>	38.41	**54.83**	2.03
2000	<u>4.21</u>	35.26	**50.56**	9.98
3000	<u>4.01</u>	33.87	**48.24**	13.89

values represent the most consumption percentages, while the underlined ones refer to the least consumption percentages.

For the evaluation operator, as shown in Tables 6 and 7, the increment of the dimension dramatically decreased the percentages of energy and time consumption. ONEMAX has an inexpensive fitness function, which made the evaluation phase has the lowest energy and time percentages (underlined) in most of the dimensions under the study. Much on the contrary, time and energy consumption percentages by the genetic operators were the highest among the other operators. The mutation operator exhibits the highest percentages (shown in bold) for all the dimensions under the study. The housekeeping tasks of the algorithm consumed small percentages of time and energy but still comparable to the evaluation operator ones.

Tables 8 and 9 present a scalability analysis for P-PEAKS problem. The boldfaced values represent the most consumption percentages, while the underlined ones refer to the least consumption percentages.

The results of P-PEAKS problem reveal the controlling rule of evaluation operator in the time and energy consumption. The evaluation operator obtained the highest consumption percentage (shown in bold) among the other operators for all the instances under the study. The evaluation requires comparing the tentative solution against a set of peaks. However, when we expand the dimension of the tentative solution, this percentage is decreased since the number of peaks is not changing, and this additional comparison has a lower impact. Crossover and mutation are applied for the solutions (of length n) and thus do not employ the number of peaks in their computation. Therefore, the increment of problem size leads to a rise in the

Table 8 Energy consumption percentages (%) of the GA components for P-PEAKS problem

Dim	Evaluation	Crossover	Mutation	Housekeeping
10	**87.22**	5.34	7.35	<u>0.09</u>
100	**86.48**	5.66	7.71	<u>0.15</u>
200	**84.76**	6.08	8.25	<u>0.91</u>
500	**75.30**	9.31	10.81	<u>4.58</u>
1000	**68.48**	12.94	14.36	<u>4.22</u>
2000	**62.71**	14.07	20.19	<u>3.03</u>
3000	**51.84**	18.09	27.09	<u>2.98</u>

Table 9 Execution time percentages (%) of the GA components for P-PEAKS problem

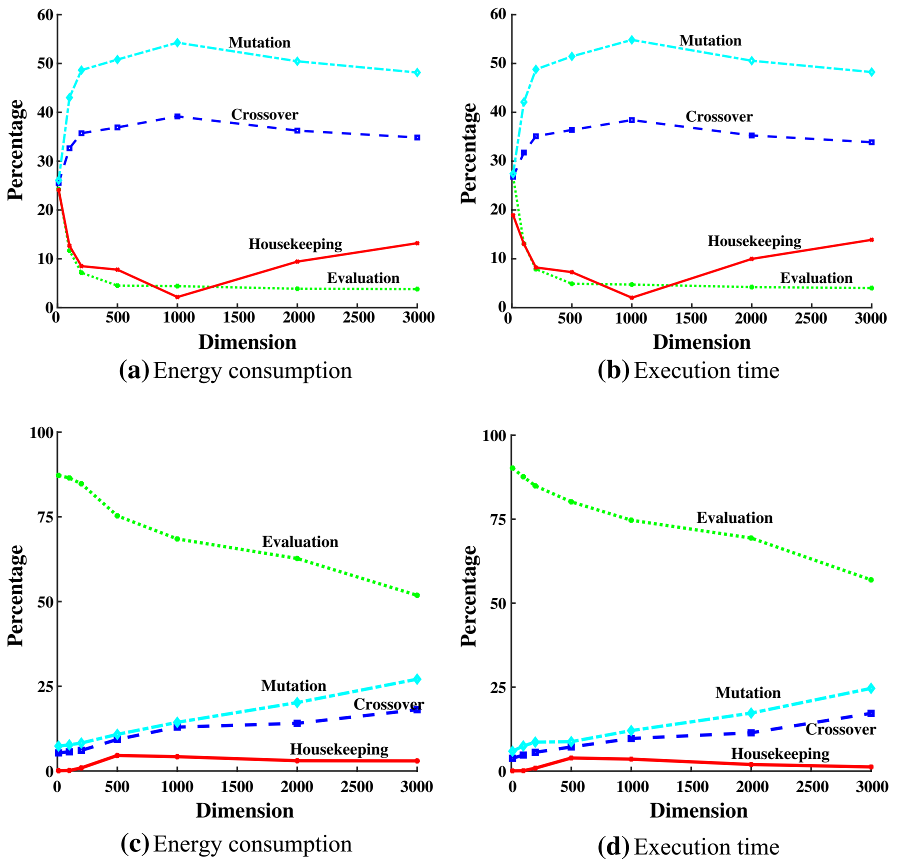
Dim	Evaluation	Crossover	Mutation	Housekeeping
10	**90.21**	3.81	5.91	<u>0.07</u>
100	**87.64**	4.77	7.48	<u>0.11</u>
200	**84.92**	5.59	8.62	<u>0.87</u>
500	**80.16**	7.18	8.76	<u>3.90</u>
1000	**74.69**	9.70	12.05	<u>3.56</u>
2000	**69.38**	11.40	17.26	<u>1.96</u>
3000	**56.94**	17.19	24.63	<u>1.24</u>

percentage of the energy consumption of the crossover and mutation. Housekeeping operator obtained the least percentage of energy and time consumption (underlined); it sustained small percentages for all the instances under the study.

For a better understanding of the energy and time percentage results shown in tables, we give a visual presentation of the results in Fig. 4.

Figure 4 shows different behaviors for the problems of different representations. For the inexpensive linear function ONEMAX, with dimension increment, we get a descent in the curve of time and energy consumption percentages for the evaluation and housekeeping tasks in contrast to a rise in the energy consumption percentage of the genetic operators. The dimension increment highly affects the genetic operations (which requires generating random numbers and swapping genes). The before-mentioned consumption behaviors are higher than the consumption of the evaluation of solutions (which is only counting bits in the ONEMAX case) and the housekeeping tasks for all the dimensions under the study.

For the nonlinear multimodal problem P-PEAKS, we have different time and energy consumption behaviors. The increment in dimension was combined with a decrement in the consumption percentages of the evolution operator. This slight decrement happens due to using a fixed number of peaks (100) for all the dimensions under study. Despite this reduction in the consumption percentages of the evaluation operator, evaluation of solutions still has higher consumption than other operators. P-PEAKS has a costly fitness function (which involves expensive computations for the distance between peaks); this reason justifies the consumption behavior of the evaluation operator.



(a) Energy consumption **(b)** Execution time

(c) Energy consumption **(d)** Execution time

Fig. 4 Energy consumption and execution time percentages versus dimension for P-PEAKS problem

Based on the previous discussions on the numerical data and figures, we can claim that enlarging problem sizes (while keeping other algorithm parameters fixed) lead to different time and energy consumptions of the algorithm operators.

5.3 Energy consumption analysis of the distributed genetic algorithm

In this section, we present an analysis of synchronous and asynchronous dGA parallel implementations. Both implementations have the same parameters with the only difference being their communication scheme (see Table 2 for the parameters used). Our algorithm consists of 32 islands for all the experiments done, and we study an increasing number of cores for running these 32 islands, from 1 to 32 cores. The results presented in this section are the basis to answer the third and fourth research questions (RQ3 and RQ4) made in the Sect. 1.

Tables 10 and 11 present the results of energy consumption and execution time of the dGA implementations, respectively. We present the results for six problem

Table 10 Mean of energy consumption values on both versions, in kWh

Problem	ONEMAX		P-PEAKS		ECC		MTTP20		MTTP100		MTTP200	
# of cores	Sync	Async	Sync	Async	Sync	Async	Sync	Async	Sync	Async	Sync	Async
1	9.94E−03	**2.40E−03**	1.34E−03	**1.83E−04**	6.80E−03	**9.25E−04**	5.57E−03	**5.47E−05**	1.39E−01	**3.09E−03**	2.95E−01	**8.11E−03**
2	2.92E−03	**1.03E−03**	3.79E−04	**8.46E−05**	1.78E−03	**3.41E−04**	1.43E−03	**2.50E−05**	4.24E−02	**1.21E−03**	7.70E−02	**3.43E−03**
4	1.42E−03	**5.19E−04**	1.80E−04	**4.73E−05**	8.81E−04	**1.73E−04**	7.46E−04	**1.85E−05**	1.91E−02	**6.35E−04**	3.29E−02	**1.85E−03**
8	5.89E−04	**1.64E−04**	8.47E−05	**1.19E−05**	3.83E−04	**4.81E−05**	2.88E−04	**7.58E−06**	8.91E−03	**2.23E−04**	1.64E−02	**5.71E−04**
16	1.94E−04	**5.55E−05**	2.65E−05	**5.08E−06**	1.17E−04	**1.75E−05**	9.06E−05	**3.14E−06**	2.44E−03	**6.03E−05**	4.99E−03	**1.68E−04**
32	**1.95E−05**	2.06E−05	2.80E−06	**2.27E−06**	7.98E−06	**6.93E−06**	2.04E−06	**1.36E−06**	2.67E−05	**2.13E−05**	7.64E−05	**6.72E−05**

Table 11 Mean of execution time on both versions, in seconds

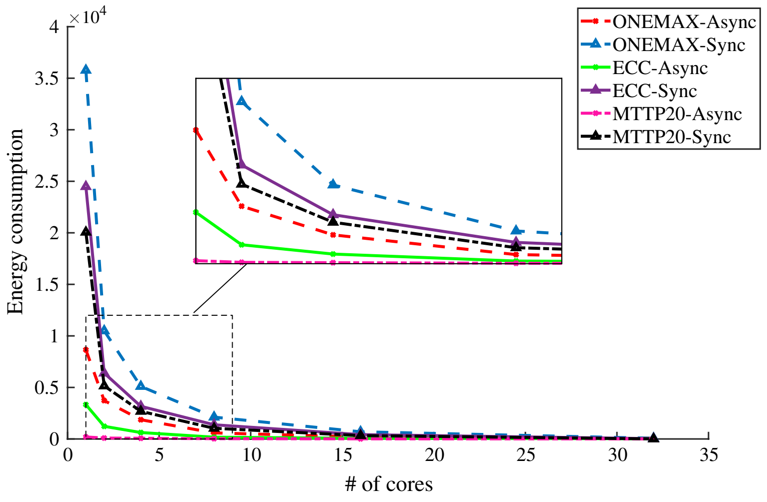
Problem	ONEMAX		P-PEAKS		ECC		MTTP20		MTTP100		MTTP200	
# of cores	Sync	Async	Sync	Async	Sync	Async	Sync	Async	Sync	Async	Sync	Async
1	48.92	**11.68**	7.09	**0.95**	33.50	**4.63**	27.64	**1.28**	740.38	**15.06**	1443.74	**39.44**
2	27.18	**9.74**	3.77	**0.86**	16.73	**3.35**	13.54	**0.57**	398.04	**11.39**	720.99	**32.38**
4	13.19	**4.91**	1.87	**0.49**	8.29	**1.72**	7.02	**0.36**	179.20	**5.95**	305.62	**17.31**
8	9.00	**2.51**	1.46	**0.25**	6.02	**0.87**	4.53	**0.25**	138.22	**3.43**	249.70	**8.49**
16	4.83	**1.47**	0.77	**0.19**	3.05	**0.50**	2.30	**0.13**	60.86	**1.63**	124.38	**4.25**
32	**0.88**	0.92	0.19	**0.17**	0.43	**0.39**	0.15	**0.10**	1.17	**0.94**	3.20	**2.83**

instances: ONEMAX of size 2000 bits, P-PEAKS of 100 bits, ECC of size 288 bits, and three instances of MTTP named MTTP20, MTTP100, and MTTP200.

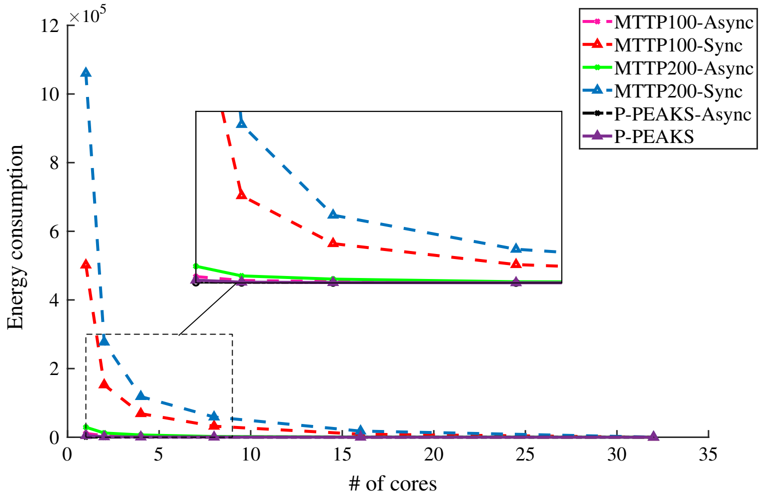
Having different problems allows raising different behaviors and consumptions, while having different dimensions for them will allow some conclusions on scalability. These results are the average of the 30 independent runs, where the stop condition was to find the optimal solution in all the cases. The results of Tables 10 and 11 clearly show that the asynchronous implementation consumes the least energy and execution times (marked in bold) in most of the instances of our experiments. These results prove that the asynchronous implementation is more energy-friendly than the synchronous implementation. Also, the larger the number of cores in the execution pool, the lower the running time and energy consumption. This result is wanted but maybe not expected, since more cores would mean more energy. Now we have quantitative evidence that parallelism can help reduce energy. In spite of the fact that adding more computing units may lead to consuming more energy, adding more computing units will steadily decrease the execution time and thus decrease the energy consumed (for the benchmark considered).

The results are rich with significant and interesting outcomes. Even if there is a big difference in dimensions between ONEMAX and MTTP200, the latter consumes a much higher amount of energy than ONEMAX for all the different numbers of cores used. This evidence confirms our previous conclusions about the importance of the evaluation operator and its complexity. On the other hand, adding more cores led to a higher reduction rate of energy and time consumption in MTTP200 more than in ONEMAX. The same energy and time reduction rates are also found on P-PEAKS and ECC problems. This behavior can be due to the larger number of cores working simultaneously to solve the problem: the higher the number of cores, the higher the number of parallel operations per time unit and therefore a smaller wall-clock time is needed. The final overall effect definitely reduces the execution time. With respect to the energy point of view, we have two different factors: When the number of cores increases, the energy consumed by the complete processor is higher, but since the execution is faster, the time in which it is consuming energy is lower. Since the second effect has a larger impact than the first one, the final energy consumption is also reduced when the number of cores is increased. With the results presented above, we can positively answer the third research question (RQ3).

Figure 5 shows the behavior of the energy consumption of synchronous and asynchronous algorithms, respectively, over a different number of cores. These results also clearly point to the fact that the asynchronous implementation is more efficient than the synchronous implementation: It consumes less energy and time in most of the cases of the study (for the same numerical performance). In terms of execution time, this behavior was reported in many previous studies on synchronism of EAs, e.g., [3, 6, 29, 40]. Our results confirm this poor performance of the synchronous implementation in terms of energy consumption too. The reason for this behavior is the unblocking of communications of the asynchronous implementation, which means there are no idle cores waiting for incoming data. Also, the independent evolution of asynchronous algorithms could promote a higher diversity inside the islands, but this effect is just clear in cluster computing with distributed memories and a network in the middle.



(a) Problems: ONEMAX, ECC, and MTTP20



(b) Problems: MTTP100, MTTP200, and P-PEAKS

Fig. 5 Energy consumption of the synchronous and asynchronous algorithms

The higher difference in energy consumption values in both versions happens when using a low number of cores (see Fig. 5). As we decrease the number of cores from 32 to 1, there will be increasingly more islands running on the same core. (Remember we do not modify the number of islands; removing or adding islands would create different algorithms and lead to meaningless conclusions.) For the synchronous implementation, this fact is more influential since fewer cores will deal with computations and synchronized communications of many islands (overhead).

The case of fewer number of cores will be more tolerated in the asynchronous implementation, since there are no waiting points and the core will not block and wait for information and computations can freely proceed.

When both implementations are being run on 32 cores (32 islands, one per core), we get a similar performance for the asynchronous and synchronous implementations: The load is similar in the two and running on a shared memory multiprocessor is equally fast for them. Also, the time cost of blocking and waiting to synchronize in the synchronous implementation is negligible. Therefore, we can totally confirm RQ3 (consumption of different communication schemes) and answer RQ4 (impact of the number of cores).

5.3.1 Statistical analysis of the synchronous and asynchronous implementations

Since we are dealing with non-deterministic algorithms, we do need a section to discuss the statistical relevance of the results. Table 12 shows a statistical comparison between asynchronous and synchronous implementations based on the energy consumption values obtained from both versions, by using the Wilcoxon's signed-rank test. Wilcoxon's test makes individual comparisons between two algorithms (pairwise comparisons) and aims to detect significant differences between them. The p-value in a pairwise comparison is independent of another one. The results are considered significant when $p < 0.01$.

The results clearly show that both versions have a different energy consumption behavior when being executed over a different number of cores. This outcome also proves our previous results in Table 10, so now we can claim that the asynchronous implementation has a different energy consumption behavior which is more efficient than the synchronous implementation.

6 Conclusions and future works

In this article, we measured and analyzed the energy and time consumption behaviors of GA and dGA, two important paradigms of optimization, search, and learning algorithms. We used different problems (varying in characteristics of the search space and fitness function complexity) over a varied number of cores and dimensions to expose the potential behavior of the algorithms. We observed that the energy consumption of problems varies according to many factors, such as the size of the problem, fitness operator complexity, and parameters used.

For the sequential GA, the fitness and genetic operators consume most of the energy and time, while the rest of the algorithm operations (housekeeping) do not take a significant amount of energy in most of the scenarios. The fitness operator of the expensive problems controls the energy and time consumption behavior of GA since other GA operations have the same implementations (not problem dependent). Mutation came out as the most consumption component in four of the problems of the study. Mutation scored higher energy consumption than crossover in all of the problems of the study. Moreover, the analysis of the relation between problem size and energy consumption reveals that the energy consumption percentage consumed

Table 12 Statistical comparison of the synchronous and asynchronous dGA, by using the Wilcoxon's signed-rank test

Problem	ONEMAX		P-PEAKS		ECC		MTTP20		MTTP100		MTTP200	
# of cores	*p* value	Sign.	*p* value	Sign.	*p* value	Sign.	*p* value	Sign.	*p* value	Sign.	*p* value	Sign.
1	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes
2	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes
4	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes
8	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes
16	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes	3.02E−11	Yes
32	8.99E−11	Yes	1.16E−07	Yes	2.00E−06	Yes	4.44E−07	Yes	6.59E−04	Yes	2.86E−03	Yes

by GA operators is varying with the change of the dimension. We remark that these percentages will not ever be the same on any laboratory experiment, but they are machine and problem settings dependent. With the previous conclusions, we could answer and justify RQ1 (energy consumption of the GA components) and RQ2 (impact of the problem dimension).

With respect to dGA, the results clearly point to a higher efficiency of the asynchronous version (time and energy), what we noticed for all numbers of cores. The statistical analysis did also confirm their different energy consumption profiles. We want to remark that the optimal energy consumption in the dGA configuration happened when using a number of islands equal to the number of cores. These conclusions conclusively give answers to RQ3 and RQ4.

For future works, we will expand our study to include other trajectory-based and population-based metaheuristics. We will analyze the energy consumption of such techniques to solve problems in smart cities domain. Finally, we plan to provide a general framework for designing efficient and energy-aware metaheuristics.

Acknowledgements This research has been partially funded by the Spanish MINECO and FEDER projects TIN2016-81766-REDT (CI-RTI), TIN2017-88213-R (6city), and Andalucía Tech, Universidad de Málaga.

References

1. Abbasi Z, Jonas M, Banerjee A et al (2013) Evolutionary green computing solutions for distributed cyber physical systems. In: Khan S, Kołodziej J, Li J, Zomaya A (eds) Evolutionary based solutions for green computing studies in computational intelligence. Springer, Berlin, pp 1–28
2. Abdelhafez A, Alba E (2017) Speed-up of synchronous and asynchronous distributed Genetic Algorithms: a first common approach on multiprocessors. In: 2017 IEEE Congress on Evolutionary Computation (CEC)
3. Alba E, Troya JM (2001) Analyzing synchronous and asynchronous parallel distributed genetic algorithms. Future Gener Comput Syst 17:451–465
4. Alba E (2005) Parallel metaheuristics: a new class of algorithms. Wiley Interscience, Hoboken
5. Alba E, Bernabé Dorronsoro (2010) Cellular genetic algorithms. Springer, New York
6. Alba E, Giacobini M, Tomassini M, Romero S (2002) Comparing synchronous and asynchronous cellular genetic algorithms. In: International Conference on Parallel Problem Solving from Nature. Springer, Berlin, Heidelberg
7. Álvarez JD, O FCDL, García Martínez JÁ, et al (2017) Estimating energy consumption in evolutionary algorithms by means of FRBS. In: Progress in Artificial Intelligence Lecture Notes in Computer Science, pp 229–240
8. Bán D, Ferenc R, Siket I et al (2018) Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware. J Supercomput 2018:1–25
9. Calandrini G, Gardel A, Bravo I et al (2013) Power measurement methods for energy efficient applications. Sensors 13:7786–7796
10. David H, Gorbatov E, Hanebutte UR, Khanaa R, Le C (2010) Rapl. In: Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design—ISLPED 10
11. Dorronsoro B, Burguillo JC, Peleteiro A, Bouvry P (2013) Evolutionary algorithms based on game theory and cellular automata with coalitions. In: Zelinka I, Snášel V, Abraham A (eds) Handbook of optimization intelligent systems reference library. Springer, Berlin, pp 481–503
12. Droste S, Jansen T, Wegener I (2000) A natural and simple function which is hard for all evolutionary algorithms. In: 2000 26th Annual Conference of the IEEE Industrial Electronics Society IECON 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation 21st Century Technologies and Industrial Opportunities (Cat No00CH37141)

13. Escamilla J, Salido MA, Giret A, Barber F (2016) A metaheuristic technique for energy-efficiency in job-shop scheduling. Knowl Eng Rev 31:475–485
14. Fanfakh A, Charr J-C, Couturier R, Giersch A (2017) Energy consumption reduction for asynchronous message-passing applications. J Supercomput 73:2369–2401
15. Goldberg D, Deb K, Horn J (1992) Massive multimodality, deception, and genetic algorithms. In: Manner R, Manderick B (eds) International Conference on Parallel Problem Solving from Nature II
16. Guzman C, Cardenas A, Agbossou K (2017) Evaluation of meta-heuristic optimization methods for home energy management applications. In: 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)
17. Hähnel M, Döbel B, Völp M, Härtig H (2012) Measuring energy consumption for short code paths using RAPL. ACM SIGMETRICS Perform Eval Rev 40:13
18. Hindle A (2016) Green software engineering: the curse of methodology. In: Proceedings: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)
19. Hooper A (2008) Green computing. Commun ACM 51(10):11–13
20. Jong KD, Potter M, Spears W (1997) Using problem generators to explore the effects of epistasis. In: The Seventh International Conference on Genetic Algorithms, pp 338–345
21. Khan KN, Ou Z, Hirki M et al (2016) How much power does your server consume? Estimating wall socket power using RAPL measurements. Comput Sci Res Dev 31:207–214
22. Khuri S, Bäck T, Heitkötter J (1994) An evolutionary approach to combinatorial optimization problems. In: 22nd Annual ACM C.S. Conference, pp 66–73
23. MacWilliams F, Sloane N (1977) The theory of error-correcting codes: part 2, vol 16. Elsevier, Amsterdam
24. Martín G, Singh DE, Marinescu M-C, Carretero J (2015) Enhancing the performance of malleable MPI applications by using performance-aware dynamic reconfiguration. Parallel Comput 46:60–77
25. Mezmaz M, Melab N, Kessaci Y et al (2011) A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. J Parallel Distrib Comput 71:1497–1508
26. Michaelides EE (2012) Environmental and ecological effects of energy production and consumption. In: Green Energy and Technology Alternative Energy Sources, pp 33–63
27. Munawer ME (2018) Human health and environmental impacts of coal combustion and post-combustion wastes. J Sustain Min 17:87–96
28. Pereira R, Couto M, Ribeiro F, et al (2017) Energy efficiency across programming languages: how do energy, time, and memory relate? In: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering—SLE 2017
29. Rada-Vilela J, Zhang M, Seah W (2013) A performance study on synchronicity and neighborhood size in particle swarm optimization. Soft Comput 17:1019–1030
30. Rauber T, Rünger G, Schwind M et al (2014) Energy measurement, modeling, and prediction for processors with frequency scaling. J Supercomput 70:1451–1476
31. Rodriguez-Gonzalo M, Singh DE, Blas JG, Carretero J (2016) Improving the energy efficiency of MPI applications by means of malleability. In: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)
32. Rotem E, Naveh A, Ananthakrishnan A et al (2012) Power-management architecture of the intel microarchitecture code-named sandy bridge. IEEE Micro 32:20–27
33. Schaffer J, Eshelman L (1991) On crossover as an evolutionary viable strategy. In: Belew R, Booker L (eds) Proceedings of the 4th ICGA, Morgan Kaufmann, pp 61–68
34. Tomassini M (2006) Spatially structured evolutionary algorithms: artificial evolution in space and time. Springer, Berlin
35. Stinson D (1985) An introduction to the design and analysis of algorithms. The Charles Babbage Research Centre, St Pierre
36. Trefethen AE, Thiyagalingam J (2013) Energy-aware software: challenges, opportunities and strategies. J Comput Sci 4:444–449
37. Tsutsui S, Fujimoto Y (1993) Forking genetic algorithm with blocking and shrinking modes. In: Forrest S (ed) 5th ICGA, Morgan Kaufmamann, pp 206–213
38. Vega FFD, Chávez F, Díaz J et al (2016) A cross-platform assessment of energy consumption in evolutionary algorithms. In: Parallel Problem Solving from Nature—PPSN XIV Lecture Notes in Computer Science, pp 548–557

39. Venkatesh A, Kandalla K, Panda DK (2013) Evaluation of energy characteristics of MPI communication primitives with RAPL. In: 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum
40. Venter G, Sobieszczanski-Sobieski J (2006) Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. J Aerosp Comput Inf Commun 3:123–137
41. Zhang H, Hoffman H (2015) A quantitative evaluation of the RAPL power control system. In: Feedback Computing