Cluster Comput
DOI 10.1007/s10586-012-0207-x

A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids

Frédéric Pinel · Bernabé Dorronsoro · Johnatan E. Pecero · Pascal Bouvry · Samee U. Khan

Received: 15 November 2011 / Accepted: 27 March 2012

Abstract The sensitivity analysis of a Cellular Genetic Algorithm (CGA) with local search is used to design a new and faster heuristic for the problem of mapping independent tasks to a distributed system (such as a computer cluster or grid) in order to minimize makespan (the time when the last task finishes). The proposed heuristic improves the previously known Min-Min heuristic. Moreover, the heuristic finds mappings of similar quality to the original CGA but in a significantly reduced runtime (1,000 faster). The proposed heuristic is evaluated across twelve different classes of scheduling instances. In addition, a proof of the energy-efficiency of the algorithm is provided. This convergence study suggests how additional energy reduction can be achieved by inserting low power computing nodes to the distributed computer system. Simulation results show that this approach reduces both energy consumption and makespan.

Keywords Task mapping · Heuristic · Sensitivity analysis · Energy-efficiency · Millicomputing

This work is supported by the Fonds National de la Recherche Luxembourg, CORE Project Green-IT (C09 IS/05).

F. Pinel (✉) · J.E. Pecero · P. Bouvry
Computer Science and Communications,
University of Luxembourg, 6, rue Coudenhove-Kalergi,
1359 Luxembourg, Luxembourg
e-mail: frederic.pinel@uni.lu

J.E. Pecero
e-mail: johnatan.pecero@uni.lu

P. Bouvry
e-mail: pascal.bouvry@uni.lu

B. Dorronsoro
Interdisciplinary Centre for Security Reliability and Trust,
University of Luxembourg, 6, rue Coudenhove-Kalergi,
1359 Luxembourg, Luxembourg
e-mail: bernabe.dorronsoro@uni.lu

S.U. Khan
NDSU-CIIT Green Computing and Communications Laboratory,
Department of Electrical and Computer Engineering,
North Dakota State University, Fargo, ND 58108-6050, USA
e-mail: samee.khan@ndsu.edu

1 Introduction

The mapping of tasks to computing resources in a distributed computing system is a challenging problem. Indeed, it is demonstrated to be NP-hard when more than two machines are considered [5]. There are many task mapping algorithms proposed in the literature to address this problem. However, because of the problem's complexity, they provide less than optimal solutions and are slow. New techniques that try to find good solutions in short computation times are needed for large scale computing systems such as Grids or Clouds that run many tasks.

The task mapping problem is further complicated with the introduction of energy minimization as an additional objective. The work presented in this paper contributes to solve this problem by proposing a new fast heuristic for the assignment of independent tasks that can be applied to the energy-efficient task mapping on distributed systems.

This paper is an extension of [31], which introduced the Two Phase Heuristic (2PH), a novel, fast and accurate algorithm as an improvement of the well-known heuristic for independent task mapping, Min-Min [16]. An interesting feature of [31] is the path that lead to such a heuristic. Indeed, the algorithm is derived from the Sensitivity Analysis (SA) performed in [28], on an elaborate Cellular Genetic Algorithm (CGA), called Parallel Asynchronous CGA (PA-CGA) [30]. PA-CGA is designed to map independent tasks on machines in a distributed system.

Published online: 27 April 2012

SA can answer the following important question: given uncertainty in system parameters, which ones affect (the most and the least) the system output (also known as *screening*) [34].

SA is useful for parameter tuning, but also at design-time. The work presented here is an example of how SA can help the development of algorithms at the design step. The results of the analysis are used to design a new heuristic to solve the independent task mapping problem.

The contributions of this paper are three-fold: (a) We first improve the study on the performance of 2PH, evaluating a new version of the algorithm that runs for a higher number of iterations and building a ranking of the algorithms according to their performance using the Friedman statistical test [14]. (b) We formally establish the convergence properties and the ability to reduce energy of the new 2PH heuristic. (c) We promote the use of millicomputers (low power but slow resources) to reduce energy in High Performance Computing Systems (HPCS), and show how millicomputing enables 2PH to reduce energy *without* increasing makespan (when the last task finishes), unlike Min-Min.

The rest of the paper is organized as follows. We summarize the existing related work in Sect. 2. Then, the considered problem is described in Sect. 3. Section 4 presents our previous work on the sensitivity analysis of the PA-CGA. Section 5 describes the new heuristic and provides a comparison with the Min-Min heuristic and the PA-CGA. Section 6 demonstrates the energy-efficiency of the mapping found, and uses the convergence study to develop a novel approach for energy-efficiency. Finally, Sect. 7 draws the main conclusions, and provides perspectives to the presented work.

2 Related approaches

A large number of heuristics have been developed to address the problem of task assignment on distributed heterogeneous systems. We begin by describing some on-line mapping heuristics. *Opportunistic Load Balancing* assigns each task, in an arbitrary order, to the next available machine, regardless of the execution time for the task on that machine. This greedy heuristic tries to balance the load on the machines, however, because the assignment algorithm does not take into consideration execution times it finds rather poor solutions. *Minimum Execution Time* assigns each task in arbitrary order to the machine with the lower execution time for that task, without considering machine's availability. The assignment algorithm intends to find good task-machine pairings, but since the heuristic does not consider the current load, this can cause load imbalance across machines. The *Minimum Completion Time* heuristic assigns each task, in arbitrary order, to the machine with the minimum expected completion time for the job. The heuristic tries to improve the performance of the previously described heuristics. This heuristic is a variant of *Limited Best Assignment* [13] and can be used as an online mapping algorithm. More details on these heuristics can be found in [4, 13].

One of the most widely used batch mode dynamic heuristic for mapping independent tasks in the heterogeneous computing system is the well-known *Min-Min* algorithm [4, 13, 16, 36]. Min-Min starts with a set of all unmapped tasks, then works in two phases. In the first phase, the algorithm establishes the minimum completion time for every unassigned task in the same way as Minimum Completion Time. In the second phase, the task with the overall minimum expected completion time is selected and assigned to the corresponding machine. The task is then removed from the set and the process is repeated until all tasks are mapped. The run time of Min-Min is $O(n^2m)$, where n is the number of tasks and m the number of machines [16]. *Max-Min* heuristic follows the same working principle as in Min-Min. The main difference is that, once the algorithm established the machine with the earliest completion time for every task, the task with the maximum earliest completion time is determined. Then the algorithm allocates the task to the corresponding machine [4]. In *sufferage* [23] the best task is the one which will be the most penalized if not allocated on its most favorable machine but on the task's second most favorable machine. The sufferage value is computed as the difference between the best minimum completion time of the task and the task's second-best minimum completion time. The method gives precedence to those tasks with high sufferage value. *Sufferage II* and *Sufferage X* are refined version of Sufferage heuristic. These heuristics no longer use as criterion the second most favorable processor but consider the first machine inducing a significant increase in the completion time [8]. *High Standard Deviation First* [26] considers the standard deviation of the expected execution time of a task as a selection criterion. The standard deviation of the execution time of a task represents the amount variation in task execution time on different machines. The task with the highest standard deviation must be assigned first for mapping. Moreover, the second part of the sufferage heuristic is applied. A set of 20 fast greedy heuristics are provided in [22]. These heuristics are founded on the idea of defining an order of task execution. For that purpose, the authors proposed task priority graph, which is constructed based on a Hasse diagram that defines a partial order between the tasks based on their Expected Time to Compute (ETC) values. The authors [12] reported three fast greedy heuristics. The heuristics are based on the list scheduling principle. First, a list of tasks is constructed based on a predefined priority. In the mapping step, the tasks are assigned to the machine that minimizes their completion time as well as their execution time. A score function is used to balance these objectives. The rational is to minimize the workload of machines.

Multiple works explored the use of metaheuristics for the task mapping problem in HPCS. In [37], a memetic algorithm is proposed. The memetic algorithm is combined with a list scheduling algorithm and several local search heuristics to find high-quality solutions and reduce execution time. A highly competitive CGA is presented in [38]. The CGA is a kind of memetic algorithm with structured population to simultaneously optimize completion time and flowtime (the sum of the tasks' finishing times). Authors in [27] report sequential and parallel evolutionary algorithms. The parallel evolutionary algorithms improves the quality of results and reduces the execution times therefore permitting to scale the problem.

3 Mapping of independent tasks problem

The problem we are investigating arises quite frequently in parameter sweep applications, such as the Monte-Carlo simulations [7]. In such a context, many tasks with almost no interdependencies are generated and submitted to the computational grid to be efficiently assigned. Efficiency means to allocate tasks as fast as possible and to optimize some criterion, such as the makespan or the flowtime. Makespan is among the most important optimization criterion of a grid system. Indeed, it is a measure of the grid system's productivity (throughput).

Task mapping is often treated as a single objective optimization problem, in which the makespan is minimized. We consider a heterogeneous computing system composed of a set $M = \{m_1, m_2, \ldots, m_m\}$ of m machines composing the computing system. We consider a set $T = \{t_1, t_2, \ldots, t_n\}$ of n independent tasks to be executed onto the system. Each task has to be processed completely on a single machine. The computational model we consider is the ETC [4], in which, it is assumed that we dispose of estimations or predictions of the computational load of each task, the computing capacity of each resource, and an estimation of the prior load of the resources. ETC allows to represent the heterogeneity among tasks and machines. We assume that the ETC matrix of size $n \times m$ is known (assumption that is made in the literature [4, 15, 18, 19]). Each position $ETC[t_i][m_j]$ in the matrix indicates the expected time to compute task t_i on machine m_j.

The scheduling problem is formulated as follows. Formally, given the heterogeneous computing systems composed of the set of m machines, and the set of t tasks. Any task is mapped without preemption from time $\sigma(t_i)$ on machine m_j, with an execution time $ETC[t_i][m_j]$. The task t_i completes at time CT_i in schedule S equals to $\sigma(t_i) + ETC[t_i][m_j]$. The objective is to minimize the maximum completion time ($C_{max} = \max(CT_i)$) or makespan.

The instance definition of the problem is as follows:

- n: the *number* of independent (user/application) *tasks* to be mapped.
- m: the *number* of heterogeneous *machine* candidates to participate in the planning.
- The *workload* of each task (in millions of instructions).
- The *computing capacity* of each machine (in *mips*).
- $ready_m$: Ready time indicating when machine m will have finished the previously assigned tasks.
- The Expected Time to Compute (*ETC*) matrix.

The two benchmark instances generated for this analysis represent different classes of ETC matrices. The classification is based on three parameters: (a) task heterogeneity, (b) machine heterogeneity, and (c) consistency [2]. In this work, instances are labelled as g_x_yyzz where:

g stands for Gamma distribution (used for generating the matrix).

x stands for the type of consistency (c for consistent, i for inconsistent, and s for semi-consistent). An ETC matrix is considered consistent if a machine m_i executes a task t faster than machine m_j, then m_i executes all tasks faster than m_j. Inconsistency means that a machine is faster for some tasks and slower for some others. An ETC matrix is considered semi-consistent if it contains a consistent submatrix.

yy indicates the heterogeneity of the tasks (hi means high, and lo means low).

zz indicates the heterogeneity of the resources (hi means high, and lo means low).

4 Sensitivity analysis of a cellular genetic algorithm

We performed, in a previous work [28], a SA on a parallel asynchronous CGA designed to map independent tasks on a distributed system, called PA-CGA [30], to look for the parameters of the algorithm that influence the most its accuracy for the given problem. This section briefly presents the results from these papers.

The chosen SA method is based on decomposing the variance of the output, as introduced by Saltelli et al. [34]. The exact implementation used is an extension to the Fourier Amplitude Sensitivity Test [35], called Fast99. Fast99 [35] allows the computation of first order effects and interactions for each parameter. Parameters interaction occurs when the effect of the parameters on the output is not a sum of their linear effects.

4.1 Parallel asynchronous cellular GA

The Genetic Algorithm (GA) analyzed is a parallel asynchronous CGA [30], called PA-CGA. PA-CGA is based on

the study reported in [29]. CGAs [1] are a kind of GA with a structured population in which individuals are spread in a two dimensional toroidal mesh and individuals are only allowed to interact with their neighbors (the set of individuals located next to the current individual in the mesh, as per some given metric). The algorithm iteratively considers as current each individual in the mesh. Consequently, parents are chosen among the neighbors with a given criterion. Following the crossover and mutation operations, the algorithm computes the fitness value of the new offspring individual (or individuals), and inserts the offspring (or one of them) in place of the current individual in the population following a given replacement policy. The cycle is repeated until termination conditions are fulfilled.

The PA-CGA algorithm is designed for multi-core processors, and therefore parallelized with threads. The population is partitioned into a number of contiguous blocks with a similar number of individuals (Fig. 1). Each block contains *pop_size*/#*threads* individuals, where #*threads* represents the number of concurrent threads executed. To preserve the exploration characteristics of the CGA, communication between individuals of different blocks is made possible. Therefore, neighborhoods may include individuals from other population blocks. Overlapping blocks allow an individual's genetic information to cross block boundaries. The different threads evolve their block independently and do not wait for the other threads to complete their generation (the evolution of all the individuals in the thread's block) before pursuing their own evolution. Therefore, if a breeding loop takes longer for an individual of a given thread, the individuals evolved by the other threads may go through more generations. The combination of a concurrent execution model with the neighborhoods crossing block boundaries leads to concurrent access to shared memory. To ensure safe concurrent memory access, we synchronize access to individuals with a POSIX [17] read-write lock. This high-level mechanism allows concurrent reads from different threads, but not concurrent reads with writes, nor concurrent writes. In the two latter cases, the operations are serialized.

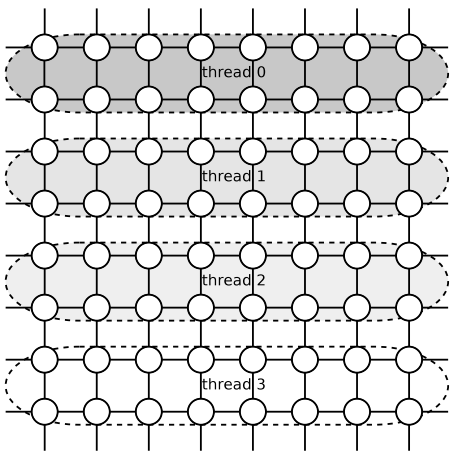


Fig. 1 Partition of an 8 × 8 population over 4 threads

Algorithm 1 Pseudo-code for one generation of an individual with PA-CGA

```
1:  while there is time left do
2:    for all ind in a thread's block do
3:      neigh := get_neighborhood(ind);
4:      parents := select(neigh);
5:      offspring := crossover(p_crossover, parents);
6:      mutate(p_mut, i_mut, offspring);
7:      HighestToLowerLoaded(H2LL)
        (p_ser, iter, offspring);
8:      evaluate(offspring);
9:      replace(ind, offspring);
10:   end for
11: end while
```

The PA-CGA (Algorithm 1) introduces a local search operator, H2LL (Algorithm 2). The operator is designed for the scheduling problem considered in this paper. The H2LL operator moves a randomly chosen task from the most loaded machine (a machine's load is the total of the tasks completion times assigned to it) to a selected candidate machine among the N least loaded ones. A candidate machine is selected if the new completion time, with the addition of the task moved, is the smallest of all the candidate machines. The move operation is performed several times (a parameter of the local search).

Algorithm 2 Pseudo-code for H2LL

```
1:  for all iterations do
2:    sort machines on ascending completion time;
3:    task := random task of the last machine in machines;
4:    best_score := CT[last machines]; {makespan}
5:    for all mac in N first machines do
6:      new_score := CT[mac] + ETC[mac][task];
7:      if new_score < best_score then
8:        best_mac := mac;
9:        best_score := new_score;
10:     end if
11:   end for
12:   move task to best_mac, if any;
13: end for
```

The following parameters have been used in the analysis of PA-CGA. The population is initialized randomly, except for one individual obtained with the Min-Min heuristic [16]. The selection operator used is binary tournament. The crossover operator used is the one-point (opx)

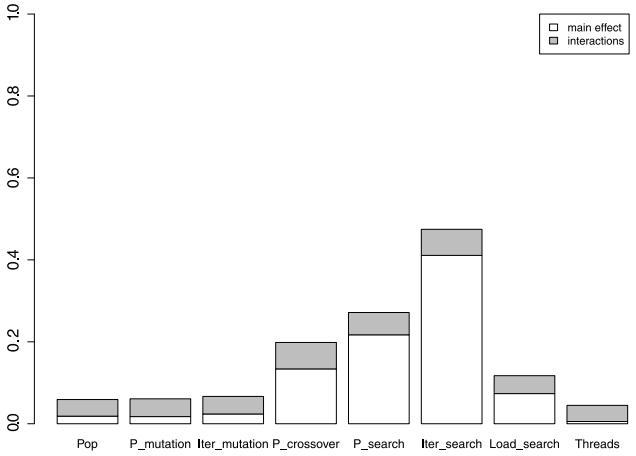


Fig. 2 Sensitivity analysis, *g_c_hihi* instance

crossover. The mutation operator moves one randomly chosen task to a randomly chosen machine. The neighborhood shape used is Linear 5 (L5), also called Von Neumann neighborhood, which is composed of the four nearest individuals (measured in Manhattan distance), plus the individual evolved. The replacement strategy is "replace if better".

4.2 Results of the sensitivity analysis

In the present study, we identified the main parameters (factors) of the algorithm to quantify their actual influence on the performance through the SA technique. Figure 2 depicts, for each of these factors, the linear and non-linear effects on the output (makespan), on problem instances with high task and resource heterogeneity, that we consider representative. The quality of the solution is defined as the average makespan over four independent runs. Performing a higher number of independent runs is extremely costly because of the high number of configurations the SA requires to test, up to 8,000. The algorithm is run for 100 generations on instances of high task and resource heterogeneity. These instances are composed of 512 tasks and 16 machines. The SA was performed for other problem instances too, and we obtained similar results.

The SA clearly shows that the local search parameters and notably the maximum number of iterations influence the output most. The number of iterations plays a role twice as big as the second most influential parameter: the local search rate. The chosen SA method is quantitative, therefore it allows such comparisons, whereas qualitative methods can only indicate the order of importance. The result of the analysis is consistent with related works in the literature that acknowledge the importance of local search in metaheuristics [25].

The results also expose that the other parameters play a limited role for the considered problem instances, *i.e.*, population size, mutation rate and mutation iterations as well as the number of threads. Such a finding is also beneficial because variables that have a positive impact on the other aspects of the algorithm, such as the algorithm's runtime can be selected without impacting the quality of the solutions. Indeed, the proposed algorithm was designed to be run for a limited period of time (wall clock); therefore choosing a smaller population size and a higher number of threads will allow the computation of more generations.

5 A two-phase heuristic

The previous section outlined the findings of the SA performed in [28]. The SA clearly showed that the specifically designed local search operator, H2LL, was very important to the quality of the mappings found. More precisely, SA also found that the number of iterations for which to perform this local search was of prime importance. The proposed Two-phase heuristic is useful because it improves on a well-known heuristic, Min-Min [16], which has been recently applied to the problem of energy-efficient mapping of tasks [20, 21, 32].

Energy efficiency is an important issue in modern-day distributed computing platforms such as grid mainly due to the required electrical power to run these systems and to cool them. This results in extremely large electricity bills, reduced system reliability and environmental issues due to carbon emissions.

Definition 1 (Energy Consumption) The total energy, E, spent by a schedule is defined by $E = \sum_{i}^{machines}(P_i \cdot CT_i)$, where CT_i is the completion time for machine i, and P_i is the power of that machine.

Definition 1 shows the relation between the completion time of machines and energy. This definition suggests that less the time spent by the system to execute the tasks, less the total energy used by the system if the power of machines does not increase. The basic strategy is to attempt to map the tasks into the fastest machines to shorten the completion time and thereby minimize energy usage. Therefore, improving the performance of Min-Min algorithm should lead to improvements in their derivative applications to energy efficient mapping.

5.1 2PH Algorithm description

The algorithm proposed, called 2PH, is simply the execution of Min-Min, followed by the local search operator H2LL, originally designed for PA-CGA.

The number of iterations for the local search in H2LL is increased from 5 to 30 and 100. The increase is motivated by the SA results, which indicated that this parameter highly influences the quality of the mappings. Additionally, the local

Table 1 Settings for the comparison with other algorithms in the literature

Parameter	Value
Instance size	128 tasks × 16 machines
Instance classes	12
Instances per class	30
Runs per instance	10
PA-CGA runtime	3 seconds
PA-CGA population	8 × 8
PA-CGA threads	1
PA-CGA mutation probability	1.0
PA-CGA mutation iteration	1
PA-CGA crossover probability	1.0
PA-CGA search iterations	5
2PH search iterations	30, 100

search is performed only once for 2PH. In contrast, PA-CGA executes H2LL for each individual in the population at each generation. Therefore, a greater number of iterations can be afforded by 2PH.

Although the 2PH algorithm is simple, there is *a priori* evidence that it should perform well. The next section describes how 2PH compares to other algorithms.

5.2 Configuration for simulations

The 2PH algorithm is first compared to Min-Min, then to PA-CGA. Wall-clock times for the 2PH and the PA-CGA implementations are useful to measure the performance of the algorithms. Moreover, mapping independent tasks is often a time-critical activity.

Our experiments were performed on an Intel Core 2 Duo CPU P8800 processor at 2.66 GHz running under Linux operating system.

Table 1 summarizes the different points of comparison for the evaluation of 2PH. A total of 360 instances were used in the comparison (30 instances of each class). PA-CGA was run for 3 seconds, wall-clock time, using 1 thread. The other parameters have identical values to those chosen for the SA. In our previous work [31], the algorithm showed similar performance for different run times ranging from 1 to 5 seconds. These times are far from those used in other previous works [30], where 90 seconds were used as time limit. However, PA-CGA with 1 thread completes over 100,000 evaluations per second of runtime, which is sufficient for the algorithm to converge to good solutions. It should be noted that PA-CGA initializes its population randomly (uniform distribution) except for one individual, which is the result of the Min-Min heuristic. Moreover, the SA shows that the number of threads does not play the biggest role in the search for good solutions.

As mentioned earlier, two versions of 2PH with 30 and 100 iterations were chosen instead of 5 for the PA-CGA. 2PH with 30 iterations only took 3 milliseconds to complete for these instances.

5.3 Simulation results

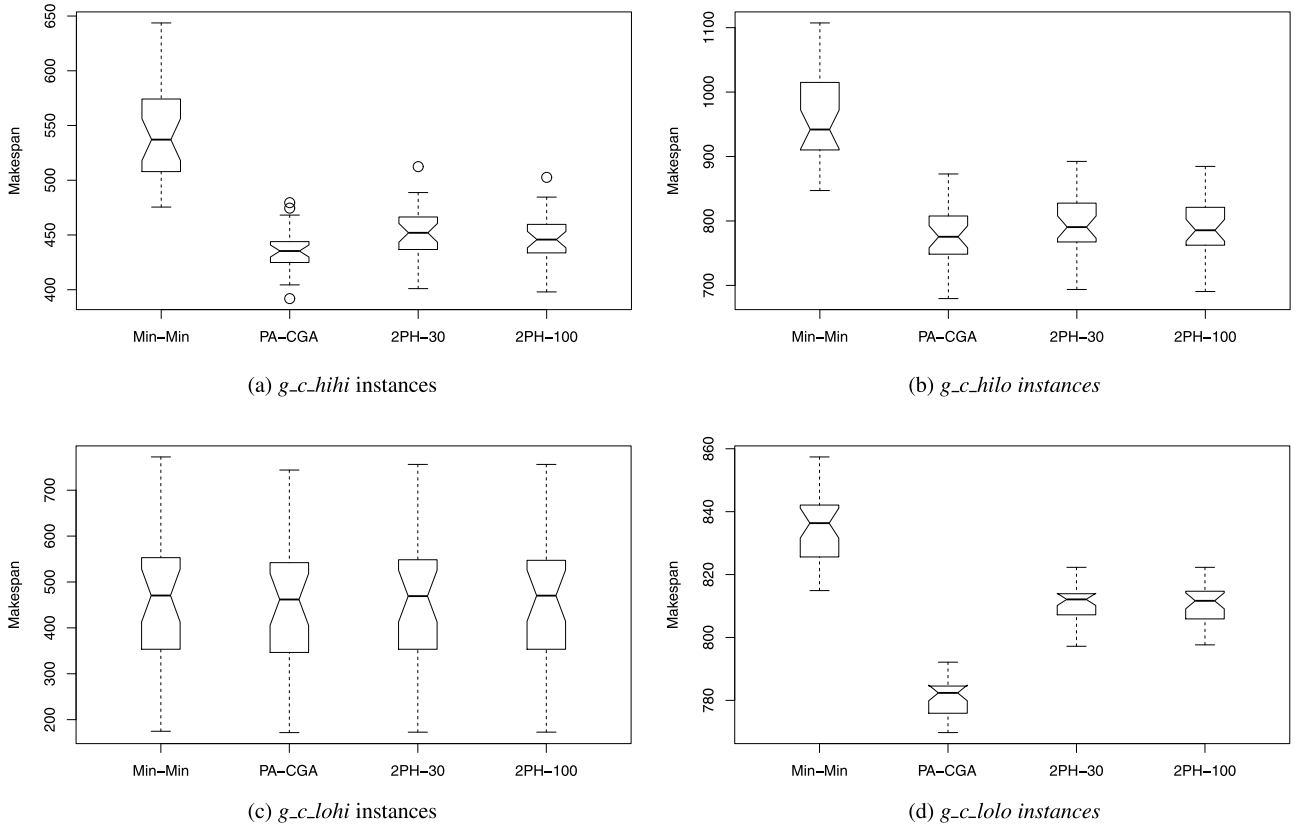
This section presents the simulation results of the different algorithms: (a) the Min-Min heuristic, (b) 2PH with 30 and 100 iterations, and (c) the PA-CGA. The results are shown as box-and-whisker plots. The boxplots are generated with the median of the makespan values obtained after the 10 independent runs for each of the 30 different instances of every problem class. The boxplots show the minimum and maximum values, as well as the first and third quartiles and the median value. The boxes with overlapping notches mean that there are not statistically significant differences (with 95 % confidence level) between the algorithms they represent.

Overall, the 2PH improves the quality of the resource allocation significantly over Min-Min, and provides results of similar quality to PA-CGA, requiring only 3 milliseconds to achieve them.

The results for the consistent, semi-consistent, and inconsistent instances are shown in Figs. 3, 4, and 5, respectively. The algorithms showed similar performance for all the problem classes. We see that there are not significant differences between 2PH with 30 and 100 iterations for any of the four problems considered with different resource and task heterogeneities. PA-CGA is the best algorithm for low task and resources heterogeneities problems, and Min-Min is always the worst one for every instance, with the exception of the instances with low task and high resources heterogeneities, for which all algorithms provide similar results. All the mentioned differences are statistically significant with 95 % confidence.

To evaluate the overall performance of the compared algorithms on all the problems, we used the Friedman statistic test to perform a ranking of the algorithms according to the solutions found. The Friedman test assigns small ranking values to those algorithms providing the highest solutions. Therefore, as the objective is minimization, those algorithms with highest rank value are the best performing ones. We computed a p-value of $1.955e{-}10$ with the Friedman test, so there are statistically significant differences with 95 % confidence on the performance of the algorithms for all the problems considered in this work.

The rank is shown in Table 2. We can see that the ranking supports our conclusions on the results. PA-CGA is the best performing algorithm, followed by the two 2PH versions (very close from each other). However, the 2PH algorithms find the solution about $1,000$ times faster than the PA-CGA, because 2PH runs for a few milliseconds versus



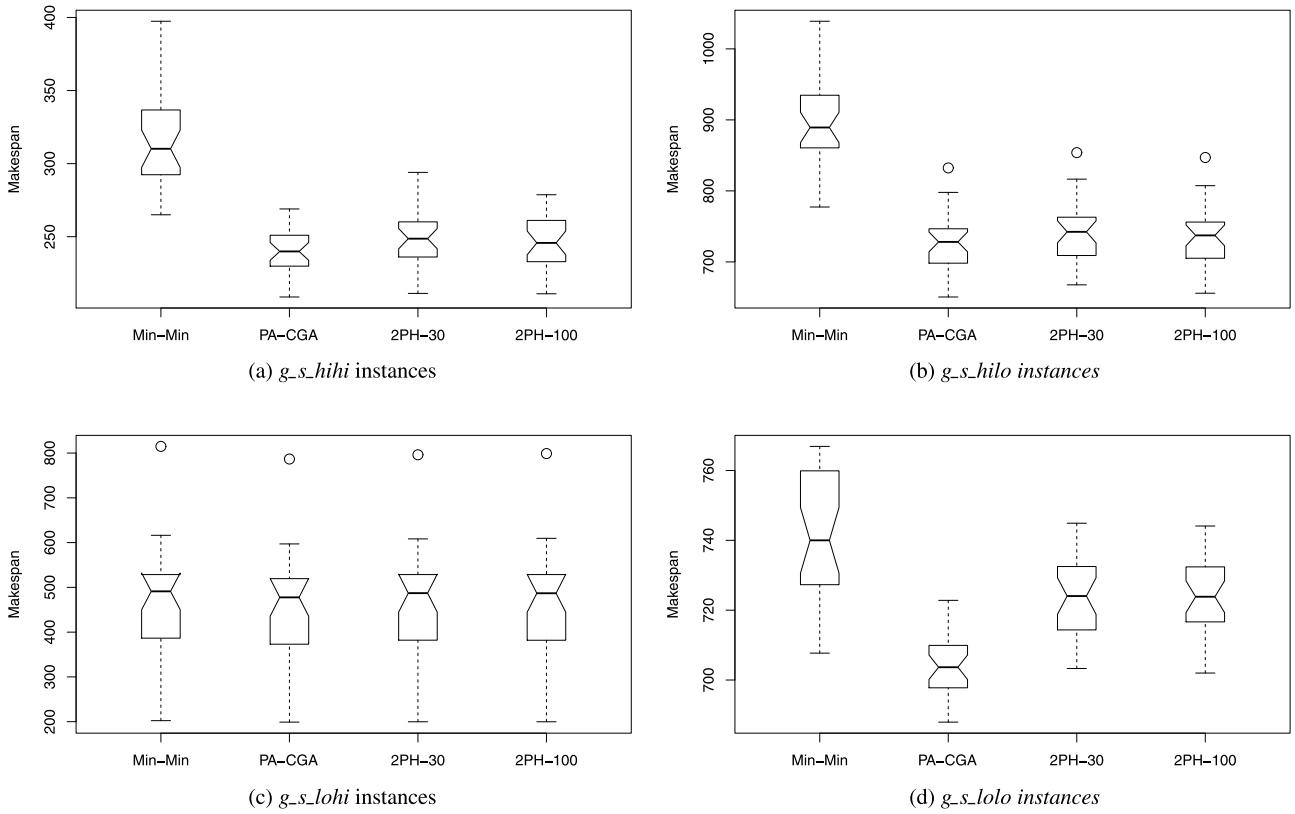
(a) *g_c_hihi* instances

(b) *g_c_hilo instances*

(c) *g_c_lohi* instances

(d) *g_c_lolo instances*

Fig. 3 Makespan for the consistent instances



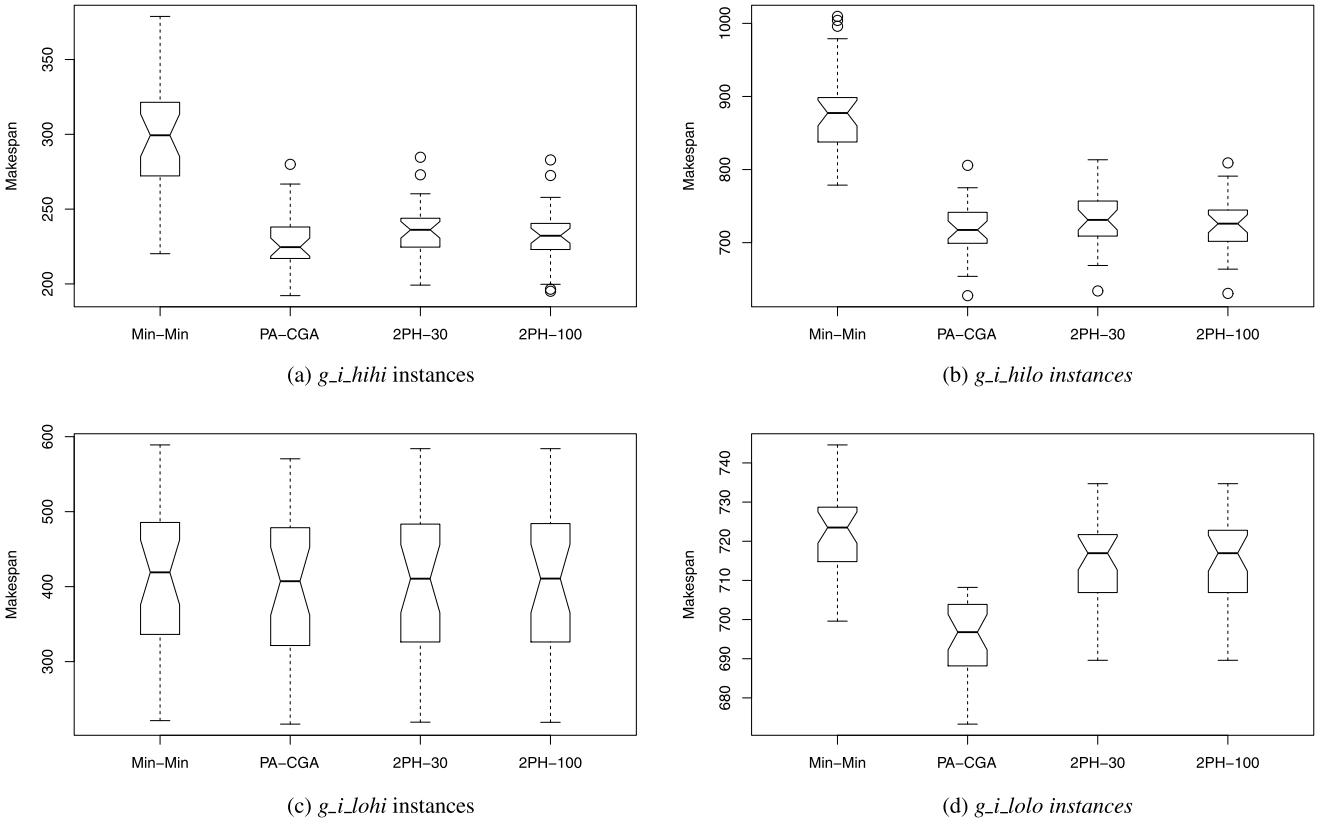
(a) *g_s_hihi* instances

(b) *g_s_hilo instances*

(c) *g_s_lohi* instances

(d) *g_s_lolo instances*

Fig. 4 Makespan for the semi-consistent instances



(a) *g_i_hihi* instances

(b) *g_i_hilo instances*

(c) *g_i_lohi* instances

(d) *g_i_lolo instances*

Fig. 5 Makespan for the inconsistent instances

Table 2 Rank of the algorithms (higher rank is better)

Algorithm	Rank
PA-CGA	3.99
2PH-100	2.725
2PH-30	2.21
Min-Min	1.079

the 3 seconds of the PA-CGA. This makes the 2PH algorithm the best option for large scale systems. Finally, Min-Min is clearly the worst algorithm of the compared ones.

6 Energy-efficiency

We demonstrate in this section that the 2PH heuristic finds energy-efficient mappings, and we later introduce a novel approach to save even more energy in large computing systems thanks to the use of millicomputers.

6.1 Energy-efficiency of the 2PH Heuristic

The main goal of 2PH algorithm is the minimization of makespan. It finds solutions to a single objective optimization problem. Although energy minimization is important, it is not considered as an additional objective function in the optimization process. The reason is that the problem becomes more complex, requiring to find a diversified set of non-dominated solutions. For this reason, multi-objective problems are normally solved with population based heuristics, that are able to generate enough diversity [10, 11]. Additionally, a multi-objective formulation would need the interaction of a decision maker to choose the most appropriate solution among the provided ones.

The question is then: how energy-efficient are the mappings found by 2PH. Sections 1 and 5 mentioned that reducing makespan also reduces the total energy spent with the mappings. This section aims to make this claim more precise.

The 2PH heuristic operates on an incremental state [33] representation (Min-Min), followed a local search (H2LL) that operates on a full state representation [33]. H2LL improves the mapping found by Min-Min. The local search can be run for an arbitrary number of iterations (depending on the runtime available to its execution). H2LL can also improve any random schedule. Therefore, the energy-efficiency of the mapping found depends on the behavior of H2LL, and not only on Min-Min (although the quality of the Min-Min mappings are well-known [4]).

After the schedule's execution, the machines are considered available to other jobs, and their energy utilization (even if idle) is not attributed to the completed schedule.

Definition 2 (Successful Transition) A transition tr, is a *successful* task move, such that $C'_{max} < C_{max}$, where C'_{max} is the actual makespan after the task movement or transition.

Theorem 1 (Completion time convergence under) H2LL *Let CT_i be the completion time of the machine i before a transition. Let $C_{max} = \max(CT_i)$, the makespan of mapping S. Let tr be a H2LL successful transition from one mapping to another. Then $\forall i, \lim_{tr\to\infty} CT_i = C_{max}$.*

Proof H2LL moves tasks from the most loaded machine to one of N less loaded machines. Let CT'_i its actual completion time after a transition. Let C_{max} and C'_{max}. By Definition 2, $C'_{max} < C_{max}$. $\forall t, \forall m_i$, either (a) $CT'_i > CT_i$, or (b) $CT'_i = CT_i$, or (c) $CT'_i < CT_i$. In (a), the machine m_i received a task moved from the most loaded machine. In (b), m_i is unchanged. In (c), the machine m_i was the most loaded and lost a task. Also, $\forall m_i$, $CT'_i \leq C'_{max}$, by definition of makespan. Therefore, in (a) and (b), CT'_i increases or remains identical, but C'_{max} decreases, therefore $\lim_{tr\to\infty} CT'_i = C'_{max}$. In (c), although CT'_i decreases, C'_{max} also decreases, therefore $\forall m_i, \lim_{tr\to\infty} CT'_i = C'_{max}$. □

Lemma 1 (H2LL reduces the energy of a balanced schedule *Let S a balanced schedule* (*that tries to minimized makespan*), *where each machine's load is approximately equal. Let E the energy of the schedule S before H2LL, and E' the energy of the schedule after H2LL. Then $E' \leq E$.*

Proof Applying Theorem 1 to the total energy, Definition 1 gives $\lim_{tr\to\infty} E = (\sum_i^{machines} P_i) \cdot C_{max}$. H2LL reduces or maintains makespan, and P_i are constants, therefore $E' \leq E$. □

Lemma 1 shows that minimizing makespan also reduces the total energy. However, there can be multiple mappings with identical makespan, for which a different assignment of tasks to machines yields a lower total energy. This is due to the heterogeneity of the power of the machines. In this case, differences in total energy between such mappings are low.

In practice, the possible gain in total energy between such mappings is small, because of (a) the relationship between power and performance of a machine, and (b) Theorem 1. Indeed, as all CT_i are of similar value, there is little available time between the completion time of a machine and makespan (this quantity is sometimes called slack). Because a machine with much lower power than another would also perform much worse, a task could not be moved from the high power machine to the low power machine without impacting makespan (the optimization objective function).

Lemma 1 considers only fairly balanced mappings, as found when minimizing makespan. In an heterogeneous cluster, loading machines that are more energy-efficient yields a low energy consumption. However, this is the opposite of makespan minimization. Inversely, balancing the load across all the machines in the cluster, including energy inefficient machines, reduces makespan but may increase energy. Using heterogeneity to lower energy is explored in the following section.

6.2 Improving energy-efficiency with millicomputing

Lemma 1 suggests how to reduce the total energy without increasing makespan. In order to reduce E, while pursuing the makespan objective, makespan must be reduced more than $\sum_i P_i$ increases. Indeed, reducing both power and makespan is not realistic, as it means obtaining better performance from lower power machines which solves the problem of energy-efficiency altogether. Given the transitions H2LL makes on the mapping (load-balancing), one approach is to increase the heterogeneity of the machines.

However, *replacing* an average machine (in terms of performance and power) with a much lower or higher power machine will not necessarily achieve the desired results. Replacing a machine with a much lower power machine may make makespan worse. Replacing a machine with a much higher power machine may worsen energy.

However, a solution is to *add* low power machines to the initial cluster of machines. Symmetrically, another solution is to add much higher power machines (with higher performance), but that is not realistic, because such machines are not available (the reason for parallel machines). The H2LL step in the 2PH heuristic balances the load of machines in the initial cluster with the new, low power, additional machines. This contributes to the makespan objective, and may not increase the $\sum_i P_i$ significantly, depending on the power specifications of the added machines. It is not necessary to add the low power machine at the Min-Min step of 2PH. Because it would not take advantage of the additional low power machines. Indeed, Min-Min incrementally builds the mapping by repeatedly assigning the task to the machine for which the completion time is minimum. The low performance of the low power machines would prevent them from being assigned any task by Min-Min.

Are such low power machines readily available? One source for heterogeneity is the alternative computer called millicomputer [9]. It originates from the rising energy costs in the data center, and suggests to replace the components responsible for the majority of the energy costs with existing, more energy-efficient and less heat producing equivalents. More precisely, the proposal is to turn to the technology in

mobile phones, smart-phones and other mobile computing devices, for solutions to the increasing cost of energy in data centers.

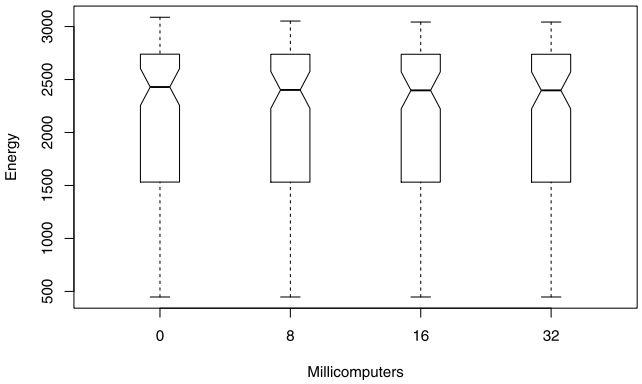
Mobile computing devices are by definition required to successfully address this issue, and it is fair to say they have partly succeeded. The mobile device industry has designed components, including processors, that can consume milliwatts (hence the name millicomputer), in contrast to the hundreds of watts of traditional servers. Furthermore, these devices do not require cooling. The processors suitable for a millicomputer cannot deliver a performance comparable to that of a typical data center processor. Therefore, it is suggested to assemble several of these energy efficient devices into a small cluster: milliclusters [9].

As an example, a millicomputer based on the ARM A8 Cortex processor (TSMC 65GP) is estimated 15× slower than a Intel 5400 series Xeon processor. This performance factor is derived from the benchmark results for the WebKit Sunspider Javascript test version 0.9.1 [24] for Intel dual core machines and a smart-phones equipped with the ARM A8 or equivalent. The Sunspider benchmark accounts for the performance of the processor but also the operating system and web browser. The performance ratio needs to be approximated because exact Sunspider benchmark results are not available for the 4 processor types used in the comparison referenced. Therefore, this ratio must be derived from the available benchmark data, where the frequency of the Intel processor is higher than the Intel 5400 series, which is our reference for the power specifications. Also, the bench-

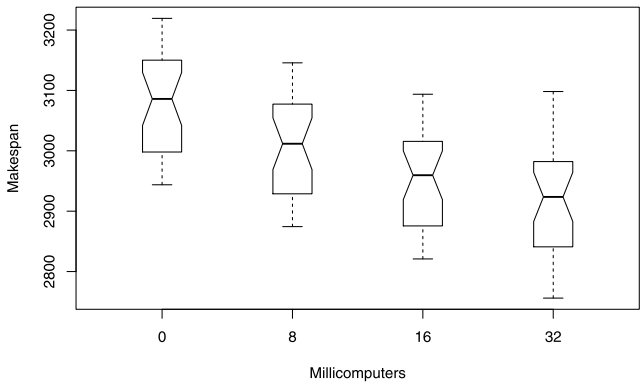
Table 3 Settings for the millicomputing simulations

Parameter	Value
Instance class	g_c_hilo
Instances per class	30
Runs per instance	30
Tasks per instance	128, 512
Cluster machines per instance	16
Power for cluster machine	200–245 W
Millicomputers per instance	0, 8, 16, 32
Power for millicomputer	5 W
Performance factor for millicomputer	15
2PH search iterations	100

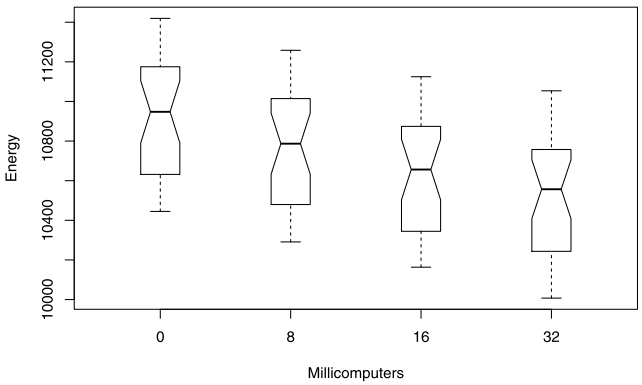
(a) Makespan for *g_c_hilo* instances of 128 tasks



(b) Energy for *g_c_hilo* instances of 128 tasks



(c) Makespan for *g_c_hilo* instances of 512 tasks



(d) Energy *g_c_hilo* instances of 512 tasks

Fig. 6 Millicomputing impact for the consistent instances

mark results for the A8 processor is based on published results from devices equipped with similar processors, such as Qualcomm's Snapdragon [3], Iphone 3 GS (which underclocks the ARM A8) and HTC Desire [6]. The power values are taken from Intel and ARM specifications documents. The ARM A8 processor consumes about 1 W, whereas the Xeon quad-core processor consumes about 40 W, at nominal activity. More recent ARM A9 MP processors are also available.

To illustrate the impact of millicomputers to a cluster, simulations are used to evaluate the makespan and energy when additional millicomputers are added to a cluster.

The settings for the simulations are presented in Table 3. Two instance sizes are used, 128 and 512 tasks. The cluster is composed of 16 machines. The cluster is considered of low heterogeneity. The power ranges from 200 to 245 W, linearly, where each increment is 3 W. The fastest machine uses the most power. A total of 8, 16, 32 millicomputers are added to the cluster. All millicomputers are considered identical. The millicomputers are considered 15 times slower than the slowest machine. The power for a millicomputer is 5 W.

Figure 6 presents the simulation results. The boxplots show the makespan and energy for the mappings found by 2PH. The boxplots show the results for the 30 instances. For each instance, the median values of the 30 independent runs is used. The x-axis indicates the number of millicomputers added. The first boxplot shows the original cluster results, without any additional millicomputer.

Two sets of instances are used, 128 tasks and 512 tasks are mapped to the cluster of 16 machines. Figures 6a, and 6b present the simulation results for instance sizes of 128 tasks. We can see very little impact of the millicomputers. This is due to the small number of tasks and the low performance of the millicomputers. If the completion time of a cluster machine is relatively low, then moving a task to a millicomputer often increases makespan. Indeed, when increasing the number of tasks, we can see a significant impact, Figs. 6c, and 6d, on both makespan and energy.

In practice, these low power machines would not necessarily be part of a cluster permanently, but placed on standby and added to the cluster on-demand, to improve both the makespan and the total energy consumption for the tasks execution.

7 Conclusions

This paper exploits the results of the sensitivity analysis of a parallel asynchronous CGA, with local search. The analysis led to the design of a simple two-phase heuristic for the mapping of independent tasks. The new 2PH heuristic was compared against two algorithms from the literature, (a) the PA-CGA, and (b) the Min-Min heuristic. In most problem instances, 2PH found equivalent mappings in much less time (milliseconds versus seconds) than the CGA. The proposed heuristic also significantly improves the mappings found by the Min-Min heuristic, with little additional computation cost. Moreover, this computational cost scales well with the problem size.

The paper presented a proof that the new heuristic also addresses the problem of energy-efficient mapping of independent tasks. Moreover, the convergence study for the heuristic provided insight which lead to a new approach to energy-efficiency in a cluster, with the introduction of millicomputing.

Future work includes the extensive experimental validation and analysis of the millicomputing alternative. We also will investigate on the performance of the algorithms for bigger problem instances.

8 Acronyms

SA	Sensitivity Analysis
CGA	Cellular Genetic Algorithm
PA-CGA	Parallel Asynchronous CGA
2PH	Two Phase Heuristic
HPCS	High Performance Computing Systems
ETC	Expected Time to Compute
GA	Genetic Algorithm
L5	Linear 5
H2LL	Highest To Lower Loaded

References

1. Alba, E., Dorronsoro, B.: Cellular Genetic Algorithms. Operations Research/Computer Science Interfaces. Springer, Heidelberg (2008)
2. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D., Ali, S.: Representing task and machine heterogeneities for heterogeneous. J. Sci. Eng. **3**, 195–207 (2000). Special 50th Anniversary Issue
3. Android 2.2: Javascript Performance. http://www.androidauthority.com/index.php/2010/07/08/android-2-2-cleans-up-with-ios4-in-javascript-performance/
4. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hengsen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
5. Brucker, P.: Scheduling Algorithms. Springer, Berlin (2004)
6. Calhoun, D.: Iphone 4 Sunspider Test Results http://davidbcalhoun.com/2010/iphone-4-sunspider-test-results/
7. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for scheduling parameter sweep applications in grid environments. In: Heterogeneous Computing Workshop, pp. 349–363 (2000)

8. Casanova, H., Zagorodnov, D., Berman, F., Legrand, A.: Heuristics for scheduling parameter sweep applications in grid environments. In: Proceedings of the 9th Heterogeneous Computing Workshop, HCW '00, pp. 349–363. IEEE Computer Society, Washington (2000)
9. Cockcroft, A.N.: Millicomputing: the coolest computers and the flashiest storage. In: Int. CMG Conference, pp. 407–414. Computer Measurement Group (2007)
10. Coello, C., Van Veldhuizen, D., Lamont, G.: Evolutionary Algorithms for Solving Multi-Objective Problems. Genetic Algorithms and Evolutionary Computation. Kluwer Academic, Dordrecht (2002)
11. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, New York (2001)
12. Diaz, C.O., Guzek, M., Pecero, J.E., Danoy, G., Bouvry, P., Khan, S.U.: Energy-aware fast scheduling heuristics in heterogeneous computing systems. In: Proceedings of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011), Istanbul, Turkey, pp. 478–484 (2011)
13. Freund, R.F., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J.D., Mirabile, F., Moore, L., Rust, B., Siegel, H.J.: Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In: Proceedings of the Seventh Heterogeneous Computing Workshop, HCW '98. IEEE Computer Society, Washington (1998)
14. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. Inf. Sci. **180**(10), 2044–2064 (2010)
15. Ghafoor, A., Yang, J.: Distributed heterogeneous supercomputing management system. IEEE Comput. **26**(6), 78–86 (1993)
16. Ibarra, O.H., Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. ACM **24**(2), 280–289 (1977)
17. IEEE and The Open Group: Posix (IEEE Std 1003.1-2008, Open Group Base Specifications Issue 7). http://www.unix.org
18. Khan, S.U., Ahmad, I.: Heuristics-based replication schemas for fast information retrieval over the internet. In: ISCA PDCS'04, pp. 278–283 (2004)
19. Khan, S.U., Ahmad, I.: Comparison and analysis of ten static heuristics-based internet data replication techniques. J. Parallel Distrib. Comput. **68**(2), 113–136 (2008)
20. Kim, J., Siegel, H.J., Maciejewski, A.A., Eigenmann, R.: Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling. IEEE Trans. Parallel Distrib. Syst. **19**(11), 1445–1457 (2008). doi:10.1109/TPDS.2008.113
21. Li, Y., Liu, Y., Qian, D.: A heuristic energy-aware scheduling algorithm for heterogeneous clusters. In: 2009 15th International Conference on Parallel and Distributed Systems (ICPADS), pp. 407–413. IEEE, Shenzhen (2009). doi:10.1007/s10586-012-0207-x
22. Luo, P., Lü, K., Shi, Z.: A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput. **67**, 695–714 (2007)
23. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings of the Eighth Heterogeneous Computing Workshop, HCW '99. IEEE Computer Society, Washington (1999)
24. Microsoft Corporation: Webkit Sunspider http://ie.microsoft.com/testdrive/benchmarks/sunspider/default.html
25. Moscato, P., Cotta, C.: A gentle introduction to memetic algorithms. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, International Series in Operations Research and Management Science, vol. 57, pp. 105–144. Springer, New York (2003)
26. Munir, E., Li, J.Z., Shi, S.F., Zou, Z.N., Rasool, Q.: A new heuristic for task scheduling in heterogeneous computing environment. J. Zhejiang Univ. Sci. A **9**, 1715–1723 (2008)
27. Nesmachnow, S., Cancela, H., Alba, E.: Heterogeneous computing scheduling with evolutionary algorithms. Soft computing—A fusion of foundations. Methodol. Appl. **15**, 685–701 (2010)
28. Pinel, F., Danoy, G., Bouvry, P.: Evolutionary algorithm parameter tuning with sensitivity analysis. In: Bouvry, P., Klopotek, M., Leprévost, F., Marciniak, M., Mykowiecka, A., Rybinski, H. (eds.) Security and Intelligent Information Systems. Lecture Notes in Computer Science, vol. 7053, pp. 204–216. Springer, Berlin (2012)
29. Pinel, F., Dorronsoro, B., Bouvry, P.: A new parallel asynchronous cellular genetic algorithm for de novo genomic sequencing. In: Proceedings of the IEEE International Conference on Soft Computing and Pattern Recognition (SOCPAR09), pp. 178–183 (2009)
30. Pinel, F., Dorronsoro, B., Bouvry, P.: A new parallel asynchronous cellular genetic algorithm for scheduling in grids. In: Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum, IPDPSW 2010, p. 206b (2010)
31. Pinel, F., Pecero, J., Bouvry, P., Khan, S.: A two-phase heuristic for the scheduling of independent tasks on computational grids. In: International Conference on High Performance Computing and Simulation (HPCS), pp. 471–477 (2011)
32. Pinel, F., Pecero, J., Khan, U.S., Bouvry, P.: Memory-aware green scheduling on multi-core processors. In: Proceedings of the Second International Workshop on Green Computing, ICPP (2010)
33. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Englewood Cliffs (2002)
34. Saltelli, A., Tarantola, S., Campolongo, F., Ratto, M.: Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models. Wiley, New York (2004)
35. Saltelli, A., Tarantola, S., Chan, K.: A quantitative, model independent method for global sensitivity analysis of model output. Technometrics **41**, 39–56 (1999)
36. Siegel, H.J., Ali, S.: Techniques for mapping tasks to machines in heterogeneous computing systems. J. Syst. Archit. **46**, 627–639 (2000)
37. Xhafa, F.: A hybrid evolutionary heuristic for job scheduling on computational grids. In: Abraham, A., Grosan, C., Ishibuchi, H. (eds.) Hybrid Evolutionary Algorithms. Studies in Computational Intelligence, vol. 75, pp. 269–311. Springer, Berlin (2007)
38. Xhafa, F., Alba, E., Dorronsoro, B., Duran, B.: Efficient batch job scheduling in grids using cellular memetic algorithms. J. Math. Model. Algorithms **7**, 217–236 (2008)

Frédéric Pinel is a Ph.D. candidate at the University of Luxembourg. He is currently working on evolutionary and machine learning algorithms applied to parallel computing and energy-efficiency. He holds Masters degrees from FUNDP, Belgium (2005) and ESIGELEC, France (1991).

Bernabé Dorronsoro received the degree in engineering (2002) and the Ph.D. in Computer Science (2007) from the University of Málaga (Spain), and he is currently working as research associate at the University of Luxembourg. His main research interests include Grid computing, ad hoc networks, the design of new efficient metaheuristics, and their application for solving complex real-world problems in the domains of logistics, telecommunications, bioinformatics, combinatorial, multiobjective, and global optimization. Among his main successful publications, he has published several articles in high impact journals and one book. Dr. Dorronsoro has been member of the organizing committees of several conferences, workshops, and special issues, and he usually serves as reviewer for leading impact journals and conferences.

Johnatan E. Pecero received a B.S. in Computer Engineering and a M.Sc. in Computer Science degree from the Instituto Tecnologico de Cd. Madero, Mexico. He earned his Ph.D. degree in Computer Science (2008) from the Grenoble Institute of Technology (formerly INPG), France. He is currently a postdoctoral researcher in Computer Science within the Interdisciplinary Laboratory on Intelligent and Adaptive Systems (ILIAS) at the Computer Science and Communication Research Unit in Faculty of Science, Technology and Communication of the University of Luxembourg. His main research interest includes Cloud, Grid and Green computing, robust scheduling, resource allocation and load balancing.

Pascal Bouvry earned his Ph.D. degree (94) in computer science at the University of Grenoble, France. He is now Professor at the Faculty of Sciences, Technology and Communication of the University of Luxembourg and heading the Computer Science and Communication research unit (http://csc.uni.lu). Pascal Bouvry is specialized in parallel and evolutionary computing. His current interest concerns the application of nature-inspired computing for solving reliability, security, and energy-efficiency problems in clouds, grids and ad hoc networks.

Samee U. Khan received a B.S. degree from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, in May 1999, and a Ph.D. degree from the University of Texas, Arlington, TX, USA, in August 2007. Currently, he is Assistant Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Moreover, he is Adjunct Professor of Computer Science and Operations Research at the North Dakota State University, Fargo, ND, USA. Furthermore, he also is an Adjunct Professor of Computer Science, COMSATS Institute of Information Technology, Pakistan. Khan's research expertise encompasses topics, such as (a) Sustainable Computing: High and low level data, task, and communications schedulers, workflow managers, and application-level dynamic fine-tuning. (b) Social Networking: Disaster management, search and rescue, classroom instructional use, and malicious behavior detection. (c) Robust Backbone Networks for Cyberinfrastructures: Resource provisioning, system recovery from catastrophic anomalies, and inter-layer/inter-domain protocols. (d) Reliability: Robustness, trust, and security. In the aforementioned areas, he has published over 125 papers. Khan is an associate editor of: (a) Cluster Computing, (b) International Journal of Communication Systems, and (c) Security and Communication Networks. He also serves on the editorial boards of: (a) Informatica, (b) Information Systems, (c) Interdisciplinary Sciences, (d) International Journal of Communication Networks and Distributed Systems, (e) International Journal of Distributed Systems and Technologies, (f) International Journal of Green Computing, (g) International Journal of Grid and Utility Computing, (h) Journal of Information Technology Research, and (i) Multiagent and Grid Systems. Khan is the recipient of the Chinese Academy of Sciences Young International Scientist Fellowship, 2011–2012, the Researcher of the Year Award, College of Engineering and Architecture, North Dakota State University, Fargo, ND, USA, 2011, the Best Paper Award of the ACM/IEEE International Conference on Green Computing and Communications (GreenCom), Hangzhou, China, December 2010, the John Steven Schuchman Memorial Outstanding Doctoral Student Award, University of Texas, Arlington, TX, USA, 2007, and the Nortel Outstanding Doctoral Dissertation Award, University of Texas, Arlington, TX, USA, 2008. He is the 2007 inductee of Upsilon Pi Epsilon, the Computer Science Honors Society. For more information, please visit: http://sameekhan.org.