A Parallel Programming Course Based on an Execution Time-Energy Consumption Optimization Problem

Javier Cuenca
Department of Engineering and Technology of Computers
University of Murcia, 30071 Murcia, Spain
Email: jcuenca@um.es

Domingo Giménez
Department of Computing and Systems
University of Murcia, 30071 Murcia, Spain
Email: domingo@um.es

***Abstract*—This paper presents an experience of Problem-based learning in a Parallel Programming course. The course includes the basics of Parallel Programming, from methodological and technological aspects to the analysis and design of parallel algorithms. The students work with an optimization problem in the field of Parallel Computing. The execution time and the energy consumption of a simplified master-slave scheme in a simplified heterogeneous system are optimized, so treating it as a bi-objective optimization problem, which is addressed with sequential, shared-memory, message-passing and hybrid parallel programming. In this way, the students follow the various parts of the syllabus of the course by working with a problem in which topics studied in previous courses are combined (green computing, computational systems architecture, optimization, heuristics), and this contributes to a deeper understanding of these topics and motivates the introduction of new concepts.**

***Index Terms*—problem-based learning; parallel programming; bi-objective optimization; execution time; energy consumption**

I. Introduction

This paper presents a problem-based learning [1] experience in which a bi-objective optimization problem is used in a Methodology of Parallel Programming course [2]. Problem-based learning is an interesting approach for computer science courses, which can be centered on the practical work of students [3], [4]. The experience presented is the continuation of a similar one [5] in which a problem-based learning approach was used on a course of Introduction to Parallel Programming, which worked on the development of parallel metaheuristics for the solution of a tasks-to-processors assignment problem. The experience was satisfactory, but the course where it was carried out has changed, with modifications in the syllabus and the location in the study plan. These modifications and recent advances in parallel computing have led us to reconsider the organization of the course, the problem used to guide teaching and the use of new tools.

The problem is presented to the students at the beginning of the course, and when the different topics of the course are studied it is used as a reference for the students to apply the techniques and tools analyzed. The problem consists of optimizing the execution time and the energy consumption of a very simple master-slave scheme in a basic heterogeneous cluster. The simulated system presents heterogeneity in computation, communication and energy consumption. The processes are assigned to the processors in the system, one process per processor, to solve the bi-objective execution time-energy consumption problem. The mapping problem is NP-hard [6], [7]. Each student proposes a solution for the mapping problem with some metaheuristic or with incomplete heuristic exact methods (exact methods where not all the possible solutions are analyzed and heuristics are used to reduce the search space). So, the students tackle some topics previously studied, for example, green computing, heterogeneous systems, optimization problems, heuristics and metaheuristic methods, and combine them in a problem which guides and motivates the introduction of new concepts in the course.

The paper is organized in the following way. Section II explains the context and the organization of the course. Section III presents the optimization problem. Section IV explains the application of some of the metaheuristics and incomplete exact methods, including their parallelization. A test is given to the students to see how the teaching objectives have been fulfilled, and the results are commented on in Section V. Finally, Section VI summarizes the conclusions and outlines possible future studies.

II. Context and Organization of the Course

In recent years parallel computing has become omnipresent, mainly propitiated by the appearance of multicores and by general purpose computing on graphic accelerators, and it is being progressively incorporated in university studies [8], [9]. The IEEE Technical Committee on Parallel Processing presented in 2012 its Curriculum on Parallel and Distributed Computing [10], which includes a list of core topics on parallelism for undergraduate studies. At the same time, to propitiate the inclusion of parallelism topics in Computational Science and Computational Engineering studies, the IEEE-TCPP launched the Early Adopter Program in 2011.

The course Methodology of Parallel Programming is part of the fourth year of the Computer Science Degree at the University of Murcia, Spain. The Degree is organized into four years, with the fourth year for specialization. There are five specializations: Computer Engineering, Software Engineering,

Computation, Information Technology and Information Systems.

A. Course Syllabus

The course lasts one semester and the topics are: Review of parallel architectures, Parallel programming paradigms, Parallel programming tools: OpenMP and MPI, Analysis of parallel algorithms, Methodology of parallel programming and Parallel algorithmic schemes. In the previous semester the students attended a course on parallel architectures, and in the following semester there is another, optional, course on Multicore Programming in the Computer Engineering specialization.

The topics of the Curriculum on Parallel and Distributed Computing [10] treated in the course are shown in Table I, where `K` represents knowledge on the topic and `C` indicates the topic is core in the course, and the students are capable of working with problems related to this topic at the end of the course. Some topics of the architecture of parallel computers are treated in the course, but they are treated more in depth in previous courses on parallel architectures. Some of the programming topics (tasks and threads, synchronization, critical regions, producer-consumer, deadlocks, etc.) are also in a previous course on principles of concurrent and distributed computing. The course is centered on parallel programming tools and environments, and especially on the analysis and design of parallel algorithms, so most of the topics on algorithms are labeled `C`, and it is expected that the students finish the course with the capacity to develop parallel algorithms and implement them on the shared-memory and message-passing paradigms.

B. Course organization

The course is compulsory for the students in the Software Engineering intensification and optional for those in the specialization of Computer Architecture. The students in the fourth year have already studied Algorithms and Data Structures, Computer Architecture, Concurrent Programming and Artificial Intelligence. A small number of students (approximately twenty-five per year) take the course, and the teaching is personalized and focused on the work of the students. They do different studies and practicals:

- Presentations on some alternative parallelism technique or tool not included in the syllabus of the course.
- OpenMP and MPI practicals with some basic problems. Tools from the Spanish Parallel Programming Contest are used [11]. The students work with some problems (problems A, B and D of the 2013 contest [12]), and send their solutions to the computational system, which evaluates them automatically and in real time. So the lecturer has access to the solutions (valid and non valid) provided by the students and detects each student's errors and corrects them in individual tutorials.
- Presentations on sequential and parallel algorithmic techniques to solve the bi-objective problem, with analysis of the possible application to their selected method of

TABLE I
TOPICS OF THE CURRICULUM ON PARALLEL AND DISTRIBUTED COMPUTING TREATED IN THE COURSE.

ARCHITECTURE	
Architecture classes	K
Streams (e.g. GPU)	K
MIMD	K
Multithreading	K
Multicore	K
Heterogeneous	K
Shared vs. distributed memory	C
SMP	K
NUMA	K
Message Passing	C
Cache organization	K
Impact memory hierarchy on software	C
Performance Metrics	K
PROGRAMMING	
Parallel Programming Paradigms	C
Shared memory	C
Distributed memory	C
Client server	K
Hybrid	K
Task/thread spawning	C
SPMD	C
Data Parallel	C
Parallel loop	C
Language extensions	C
Compiler Directives pragmas	C
Libraries	C
SPMD Notations	C
MPI	C
CUDA/OpenCL	K
Task and Threads	C
Synchronization	C
Critical regions	C
Producer-consumer	C
Deadlocks	C
Memory models	K
Scheduling and computation	C
Decomposition strategies	C
Scheduling and mapping	K
Data Distribution	K
Performance monitoring	K
Performance metrics	C
Speed-up	C
Efficiency	C
Amdahl's law	K
Isoefficiency	C
ALGORITHMS	
Asymptotics cost	C
Time	C
Space	C
Speedup	C
Cost	C
Divide and Conquer	C
Algorithmic Problems	C
Matrix computations	K
Termination detection	C
Dependencies	C
Broadcast	C
Asynchrony	C
Synchronization	C
Sorting	C
Graph search	C
Specialized computations	C
CROSS CUTTING	
Why and what is PDC	K
Power	K
Cluster, Cloud, Grid	K

the parallel programming methodologies and algorithmic schemes in the syllabus of the course.
- Development of an algorithm to solve a challenging problem sequentially (in our case the bi-objective time execution-energy consumption problem), and development of parallel versions (for shared memory with OpenMP, for message-passing with MPI, and with MPI+OpenMP hybrid parallelism) of the sequential algorithm. This part has the highest value in the final mark, and the course is oriented to accomplish this work successfully.

The use of a challenging problem to guide teaching is an interesting approach, and has been used to teach both parallel programming [13] or optimization techniques [14]. In our course, each student will develop sequential and parallel algorithms for the solution of a challenging problem. The proposed problem is a mapping problem where p processes on a master-slave set (one master and $p-1$ slaves) are assigned to p processors in a heterogeneous system, each process to a different processor. Solutions with low execution time and energy consumption are sought. The master-slave scheme considered is a simple one, but the heterogeneity of the system and the bi-objective problem stated make it rich enough for this course. So, the students tackle a challenging problem in the field of parallel computing, and they work with topics studied on previous courses.

The methods proposed to solve the problem are of three types:

- Exact methods, for example, Backtracking and Branch and Bound, combined with Greedy algorithms. Due to the high computational cost of the problem and the sizes to work with, exhaustive search in the solutions space is not viable, and some pruning strategy must be included, with possible pruning of branches with optimum solutions. So, the exact methods we consider are incomplete in the sense that not all the possible solutions are exploited, and consequently we do not know if the solutions given are the best.
- Distributed metaheuristics, like Scatter search, Genetic algorithms, Ant colony and Particle swarm optimization. These metaheuristics work with populations or sets of elements which are combined in some way to improve the elements in the set and the best solution, or the Pareto front in the bi-objective problem.
- Neighborhood metaheuristics, like Hill climbing, Tabu search, Guided local search, Variable neighborhood search, Simulated annealing and GRASP. These metaheuristic methods work with one element in the solution space and search for better elements in its neighborhood.

There are many books on algorithms [15], [16] and metaheuristics [17], [18] which can be consulted by the students to adapt the general scheme to the problem proposed. Additionally, each student should look for parallel strategies for the method they selected [19]. Each student prepares a presentation on the general ideas of the technique chosen and on strategies for parallelization with OpenMP, MPI and MPI+OpenMP. The presentations come before the practical work, so the students can share ideas about some parts of the problem (the representation of solutions and nodes, the general scheme of the algorithms, schemes of metaheuristics, possible combinations of techniques, etc). Student collaboration is therefore fostered. The experimental comparison of the different techniques developed by the students is valued in the final evaluation of the practical. Additionally, at least two individual tutorials with each student are organized, one before and one after the presentations.

III. The Optimization Problem

The simple master-slave scheme considered is shown in Algorithm 1. There is a total of p processes, a master process sends identical tasks to $p-1$ slaves in the order $1, 2, \ldots, p-1$ and waits for the solutions in the same order. Each slave receives one task, solves it, and sends the solution back to the master. The simulated computational system consists of p processors connected through an interconnection network. The system is heterogeneous in communication, computation and energy consumption, and the costs vary depending on the parts of the algorithm and the processor where they are carried out or the source and target processors involved in a communication. The algorithm consists of three parts: the send-receive of the tasks, the computation of the tasks, and the send-receive of the solutions. There are several ways to model the execution time and the energy consumption [20], and in the proposed problem the characteristics of the computational system are determined with values representing the costs in the different parts of the algorithm. Several tables store the simulated computational and communication costs:

Algorithm 1 Basic master-slave scheme used for the bi-objective optimization problem

```
IN PARALLEL in each process P_i (i = 0, ..., p − 1) DO
if i = 0 then
  for j = 1 TO j = p − 1 do
    Send task to P_j
  end for
  for j = 1 TO j = p − 1 do
    Receive solution from P_j
  end for
else
  Receive task from P_0
  Solve task
  Send solution to P_0
end if
END PARALLEL
```

- A table $TimeComunT$, of size $p \times p$, stores the costs of communications between two processors when sending-receiving a task. $TimeComunT_{ij}$ is the cost of the sending when the source process is in processor i and the target process in processor j.

- A table of the same size, $TimeComunS$, stores the costs of communications between two processors when sending-receiving a solution.
- The computation costs for the solution of the tasks are stored in an array $TimeCompuT$, of size p, with $TimeCompuT_i$ the cost of solving a task by a process in processor i.
- The energy consumption costs are stored in four arrays of size p, which store the energy consumption per time unit in the four parts in which a processor can be during the application of the algorithm:
  - The energy consumption on each processor working in the communication of a task ($EnerComunT$).
  - The consumption of processors in the communication of solutions ($EnerComunS$).
  - The consumption in a processor when working on the solution of the task ($EnerCompuT$).
  - The consumption when the processor is idle ($EnerIdle$). A processor can be idle because the process assigned to it has not begun the execution or because it is waiting for a message.

To clarify the meaning of the different costs and how the total execution time and energy consumption are modeled, let us consider a small example with $p = 4$. Tables II and III show values for $TimeComunT$ and $TimeComunS$. In $TimeComunT$ and $TimeComunS$ the positions of source and target processors have been exchanged. Values for the task computation time and for the energy consumption in the different execution states of the processors are shown in Table IV. The times when communicating tasks and solutions (Tables II and III) are different and change between different pairs of processors, so the system is heterogeneous in communications, heterogeneity in the application of different communications is considered, and the direction of the communication also contributes to the heterogeneity. Furthermore, the cost of communications is lower than that of computations (row $TimeCompuT$ in Table IV). Heterogeneity in the energy consumption (Table IV) is also included, with variations depending on the processor and the execution phase, and the lowest values are considered for idle periods, the highest for the computation phase, with medium values (but much lower than those for the computation) considered for the communications.

TABLE II
EXAMPLE OF A TABLE $TimeComunT$ OF EXECUTION TIMES OF THE COMMUNICATION OF THE TASKS.

Source/Target	0	1	2	3
0	1	1	2	2
1	2	1	3	2
2	2	3	2	2
3	1	2	3	2

The students must consider the general problem, with any values in the tables, but these values can also be established so that they correspond to real situations. For example, a master-slave matrix multiplication could be considered, with square matrices of size $n \times n$, with $\frac{n}{p-1}$ rows of the resulting matrix to be computed by each slave process; so the cost of each communication is modeled $t_s(i,j) + \frac{n^2}{p-1} t_w(i,j)$, with $t_s(i,j)$ the start-up time from processor i to processor j, and $t_w(i,j)$ the corresponding word-sending time; and the cost of each task is $2t_c(i)\frac{n^3}{p-1}$, with $t_c(i)$ the cost of a basic floating point operation in processor i. Concerning energy consumption, the mean cost of each phase of the algorithm in each processor can be obtained experimentally.

TABLE III
EXAMPLE OF A TABLE $TimeComunS$ OF EXECUTION TIMES OF THE COMMUNICATION OF THE SOLUTIONS.

Target/Source	0	1	2	3
0	1	2	1	2
1	1	2	2	2
2	2	2	3	2
3	2	3	3	3

TABLE IV
EXAMPLES OF THE VECTORS REPRESENTING THE EXECUTION TIMES OF THE SOLUTION OF THE TASKS AND ENERGY CONSUMPTIONS OF THE DIFFERENT PARTS OF THE ALGORITHM.

Processor:	0	1	2	3
$TimeCompuT$	10	8	9	12
$EnerComunT$	10	12	8	14
$EnerComunS$	12	14	10	14
$EnerCompuT$	100	80	90	110
$EnerIdle$	4	7	5	2

The problem consists of obtaining one-to-one process-to-processor assignments which give satisfactory modeled execution times and energy consumption. So, we have a bi-objective problem and pairs at the Pareto front are searched for [21]: a pair (t_1, e_1) is discarded when there is another pair (t_2, e_2) with $t_2 \leq t_1$ and $e_2 \leq e_1$. A mapping is determined by a permutation $\pi = (\pi_0, \pi_1, \ldots, \pi_{p-1})$ of $(0, 1, \ldots, p-1)$, where π_i represents the processor to which process i is assigned. Given a mapping π, the timeline of the execution of the algorithm in the particular system (the p processors with the tables of time and energy) can be simulated, and the modeled execution time and energy consumption corresponding to π are computed. For example, for $\pi = (3, 2, 0, 1)$, which means the master process is assigned to processor 3, the first slave to processor 2, the second to 0, and the third to 1, the timeline of the execution is presented in Figure 1. The light gray rectangles (left) correspond to communications of tasks, the communication of solutions are represented in dark gray (right), the computation in black, and the white areas indicate phases without activity. The modeled execution time is the final time, $t(\pi) = 20$, and the energy consumption is calculated by adding up the energy consumed by the four processors, $e(\pi) = \sum_{i=0}^{3} e_i$, with $e_i = t_{ComunT} EnerComunT(i) + t_{ComunS} EnerComunS(i) + t_{Compu}(i) EnerCompT(i) + t_{Idle}(i) EnerIdle(i)$, and $t_{ComunT}(i)$, $t_{ComunS}(i)$, $t_{Compu}(i)$ and $t_{Idle}(i)$ represent the time processor i is receiving the task,

sending the solution, working on the task and being idle. The energy consumed by processor 0 is $e_0 = 1*10+2*12+10*100+7*4 = 1062$, and calculating in the same way the energy in the other processors, the total energy is $e(\pi) = 2910$.

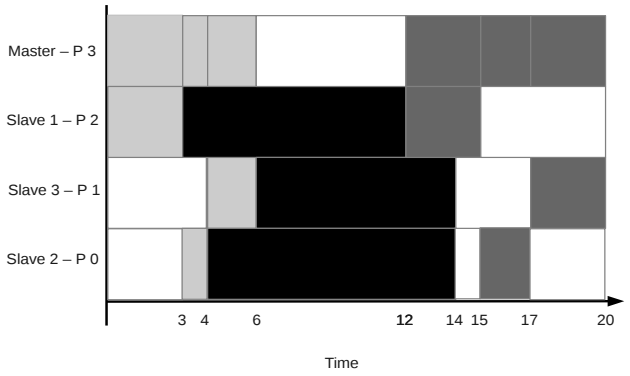

Fig. 1. Timeline of the execution for the example with 4 processes, assignment $\pi = (3, 2, 0, 1)$ and execution times and energy consumption costs in tables II, III and IV.

The problem becomes a tree traversal problem if we consider the tree of all the possible mappings. Because we are searching in the permutations space, the tree is permutational, with $p!$ terminal nodes, each corresponding to a possible mapping. For large problems (large numbers of processes and processors) the problem can not be solved by generating all the configurations, simulating the execution line for each of them, and storing those whose pair execution time-energy consumption is at the Pareto front. Configurations with p ranging from 10 to 40 are provided to the students, who are asked to obtain solutions and to study the execution time and goodness of the solution for these configurations. Additionally, they can experiment and study the behavior of their programs with other configurations, but the use of these common configurations facilitates the comparison of the different solutions the students obtain. Furthermore, some students can work together in the experimentation and comparison of the methods they develop, which can then be considered for the final mark of their work. The largest configurations can not be tackled with exact methods, so they must use non-exact methods or incomplete exact methods, and a small number of students work with each method from the list mentioned in Section II.

IV. Application of Metaheuristics and Incomplete Exact Methods

This section comments on some of the methods proposed to the students for application to the bi-objective problem presented. The methods proposed to the students can be classified in three groups, and we comment on the characteristics and possible works to be done with one method from each group. A Backtracking with node pruning is selected from heuristic incomplete exact methods; Genetic algorithm is possibly the most popular metaheuristic, and it is the distributed metaheuristic selected; and Tabu search is chosen as representative of metaheuristics based on neighborhood search. The students work on the solution of the bi-objective problem, but the course is on Parallel Programming, so they are not asked to devote too much effort to the optimization of the sequential solutions, but to develop simple sequential versions with the characteristics of the method they work with, and to work with OpenMP, MPI and MPI+OpenMP in the parallelization of the sequential versions.

A. Incomplete Backtracking

Backtracking methods can be used for processes-to-processors assignment problems. For the simulation of small systems, backtracking is satisfactory, but for large systems huge assignation times are needed. So, the work of the students is:

- For the sequential method:
  - To understand the mapping problem and the increment of the assignation time when backtracking is used for large systems, which makes the backtracking impractical in most cases.
  - To develop a backtracking scheme for the proposed assignment problem. The scheme should include a pruning routine which should be easy to substitute to experiment with different pruning techniques.
  - To identify possible techniques to eliminate nodes, which in some of the cases would not lead to the optimum mapping.

    An established maximum number of descendants is considered for each non terminal node in the solutions tree. The nodes to explore are those with the highest estimation. Because the goodness of the solution depends on two objectives, a joint indicator can be used (for example the multiplication of the estimated execution time and energy consumption) or a part of the nodes can be selected on the basis of one objective and the rest of the nodes on the basis of the other. It is necessary to determine satisfactory estimations which guide the search and contribute to the non elimination of the branches where the best solutions are. Normally a low execution time implies low energy consumption, and so the estimation can be based on the execution time.

    To limit the number of descendants of a node may not be sufficient to reduce the tree to an affordable size; so, from a certain level on only one descendant could be considered, which means a greedy approach.

    A minimum execution time and energy consumption can be associated to a node with a partial assignment. If the processor where the master process is assigned is selected at the first level, this level could be considered different, by including all the nodes in the pruned tree. Successive levels would represent

the assignation of slaves, beginning from slave 1 to slave $p-1$, so the execution time and energy consumption (pair (t, e)) of an algorithm with only the first s slaves (in a node at level $s+1$) would be lower bound for the optimum values achievable from this node, and the pair can be compared with the pairs in the current Pareto front, and if the pair is discarded, so would the optimum solution from the node, which can be pruned.
A complete assignment from the partial assignment represented by the node can be obtained. For example, if the partial assignment is $\pi = (0, 3, -, -)$ (the master is assigned to processor 0, the first slave to processor 3, and nothing has been decided for the second and third slaves), the remaining slaves can be assigned in different ways and the time for the assignment obtained would represent the estimation of the node. Some simple assignment of the slaves can be considered, for example assigning the remaining processes consecutively ($\pi = (0, 3, 1, 2)$), or some heuristic method can be considered, for example, to assign first to the processors where the computation is most costly.
  - A complete backtracking would be applied to small problems, and the results obtained are compared with those when the incomplete backtracking is applied. Once the incomplete backtracking gives satisfactory results for small problems, experiments for larger problems are carried out, and modifications are included in the pruning process to reduce the number of nodes and consequently the execution time of the backtracking.

- Different schemes can be considered to obtain parallel versions. With a master-slave scheme the work could consist of:
  - With OpenMP, the master generates nodes up to a certain level, starts the slaves, and all these do backtracking from the nodes dynamically assigned to them. The values of the best solutions can be shared by the threads so that they can be used in the pruning.
  - The MPI version works in the same way, but in this case the master processor sends nodes to the slave processors and these send back the results to the master. Sharing the information about the best solutions means communications, and the influence of the frequency of communications in the execution time and the Pareto front generated should be experimentally analyzed.
  - A hybrid MPI+OpenMP version can be easily obtained from the two previous versions, with static cyclic distribution between processes and dynamic assignation to the threads managed by each process.
  - The sequential and parallel versions should be compared. The speed-up and scalability of the parallel versions are analyzed, as is the influence of the pruning and the sharing of information between processes in the execution time and the Pareto front generated.

B. Genetic Algorithm

Genetic algorithms are possibly the most popular metaheuristic techniques [22]. The students had already seen this technique in a course on Artificial Intelligence. The work with this technique would be:

- For the sequential method:
  - To understand the mapping problem and to identify population and individual representations, to identify the possible forms of the basic routines in the Genetic algorithm, and to develop a genetic scheme at a high level. The scheme must allow easy changing of some parameters (i.e., number of individuals in the population and number of iterations to converge) or routines (i.e., mutation, combination, etc).
  - To experimentally tune the values of the parameters and the routines (modifying them in the high level scheme) to the assignment problem. In general, to obtain low assignation times it is necessary to reduce the number of individuals and the number of iterations, but this reduction does produce a reduction in the goodness of the solutions.
- About the parallel versions:
  - The OpenMP program can work by simply parallelizing the combination of the population, which includes the estimation of execution time and energy consumption associated to the generated individuals.
  - Of the parallel genetic schemes [23], the island scheme is the most appropriate for the message-passing version. The number of generations to exchange information between the islands is one of the parameters that has to be tuned.
  - An island scheme with threads working in the computations at each island is a hybrid approach.
  - The execution time of sequential and parallel versions should be compared, analyzing speed-up and scalability, and also the influence of some parameters (number of islands, size of generations and migrations, etc) in the Pareto front.

C. Tabu Search

Tabu search is a local search technique which uses memory structures to guide the search. The students had already seen this technique in a course on Artificial Intelligence. The work could consist of:

- For the sequential method:
  - Understand the assignment problem and identify set and element representations, and identify how the basic routines in the Tabu search would be for the bi-objective problem. The scheme must allow easy change of some parameters (i.e., number of iterations a movement is tabu, feasibility value of a

stage depending on the frequency of the movement, number of iterations to converge, etc) or routines (i.e., generation of the first stage).
  - Experimentally tune the values of the parameters and the routines (modifying them in the high level scheme) to the assignment problem.
- About the parallel versions:
  - The OpenMP program could work by selecting a number of nodes to explore at each step equal to the number of threads, or several independent tabu searches could be conducted in different threads.
  - For the MPI version, different tabu techniques can be analyzed [24]. One simple and effective method could work with each process performing independent search, with multiple initial solutions and different search strategies, and without knowledge sharing between processes.
  - A MPI+OpenMP version is easily obtained, with independent search in each thread.
  - The sequential and parallel versions should be compared, but in this case the comparison of execution time is not important because the parallelism is used for further search, which could give better Pareto fronts.

V. Evaluating Teaching

In order to assess whether the teaching objectives had been fulfilled, a questionnaire was given to the students. Some of the statements in the test are about the general organization of the course (statements 1 to 3), others about the interest of the bi-objective problem proposed to clarify ideas of some topics previously studied (4 to 7), and others about the utility of the problem for parallel programming learning (8 to 10). Each item is valued from 1 to 5, with 1 meaning total disagreement and 5 total agreement. Table V shows the statements in the test and summarizes the answers. All the items have high values, which indicates the perception of the students of the organization of the course is positive. We comment on some aspects of the answers to the test:

- The students found the general organization of the course appropriate: the mean of the answers for questions 1 to 3 is 4.5, and question 8 (about the use of a challenging problem to guide the course) was scored 4.3.
- The mean answer for questions 4 to 7 is 4.2, so it seems the problem selected to guide the course was useful for a deeper learning of concepts previously studied. The highest score is obtained in question 6 (about bi-objective optimization) and the lowest in question 4 (about energy aware algorithms), possibly because the students did not have previously worked with bi-objective problems, and because the algorithmic scheme to be optimized is a very rough simplification.
- Working with a challenging problem was seen as interesting (question 8), and in particular with the bi-objective problem stated (question 9), so the use of the problem-guided learning strategy is useful for this type of courses.
- From the answers to question 10, we see the students consider the organization of the course has not supposed an important increase in their work load, or if it has, compared than with other methodologies, this was compensated by the active learning approach.

VI. Conclusions and Possible Future Studies

The paper presents a teaching experience of the use of a challenging problem in the field of parallel computing to guide the teaching in a parallel programming course. The problem is given to the students at the beginning of the course, and each student works on the solution of the problem with a different method. The problem proposed includes topics previously studied, so fostering a better understanding of them and putting them into practice. It is a bi-objective mapping problem, where the execution time and the energy consumption of a master-slave scheme in a heterogeneous system must be optimized. Thus, the topics the students review are: green computing, heterogeneous systems, optimization, bi-objective problems, metaheuristics and heuristics in exact methods. Because all the students work with the same mapping problem, but each student works with a different method, collaboration between students and common enrichment is fostered.

The success of the course organization was evaluated through a questionnaire for the students. The experience seems to be very positive, and so it will be continued in successive courses. Other mapping problems in the field of parallel programming or other challenging problems in scientific or engineering can be considered, for example DNA analysis or molecules simulation.

Acknowledgment

This work was supported by the Spanish MINECO, and by European Commission FEDER funds, under grant TIN2015-66972-C5-3-R.

References

[1] M. J. O'Grady, "Practical problem-based learning in computing education," *Trans. Comput. Educ.*, vol. 12, no. 3, pp. 10:1–10:16, Jul. 2012. [Online]. Available: http://doi.acm.org/10.1145/2275597.2275599

[2] D. Giménez, Web page of the Methodology of Parallel Programming course at the University of Murcia, `http://dis.um.es/~domingo/app.html`.

[3] E. Nuutila, S. Törmä, and L. Malmi, "PBL and Computer Programming - The Seven Steps Method with Adaptations," *Computer Science Education*, vol. 15, no. 2, pp. 123–142, 2005.

[4] R. S. Pinto, P. N. Nobile, E. L. C. Mamani, L. P. Júnior, H. J. F. Luz, and F. J. Monaco, "Operating System from the Scratch: a Problem-Based Learning Approach for the Emerging Demands on OS Development," in *ICCS*, 2013, pp. 2472–2481.

[5] A.-L. Calvo, A. Cortés, D. Giménez, and C. Pozuelo, "Using metaheuristics in a parallel computing course." in *ICCS (2)*, ser. Lecture Notes in Computer Science, M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5102. Springer, 2008, pp. 659–668. [Online]. Available: http://dblp.uni-trier.de/db/conf/iccS/iccS2008-2.html#CalvoCGP08

[6] H. Lennerstad and L. Lundberg, "Optimal scheduling results for parallel computing," *SIAM News*, pp. 16–18, 1994.

[7] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 4, pp. 319–330, 2004.

TABLE V
STATEMENTS OF THE QUESTIONNAIRE FOR THE STUDENTS, AND MEAN ANSWERS.

Statement	Mean
1. The session on presentations of parallel techniques and tools is interesting and should be maintained for future courses	4.5
2. The use of problems from the Spanish Parallel Programming Contest and of its evaluation tool helps to learn basic concepts of parallel programming	4.5
3. The session on presentation of the different methods and their parallelization is interesting and allows comparison of different approaches	4.6
4. The mapping problem has helped to understand energy aware algorithms better	3.8
5. The mapping problem has helped to understand heterogeneous systems better	4.1
6. The mapping problem has helped to understand optimization and bi-objective problems better	4.8
7. The mapping problem has helped to understand some metaheuristics and heuristics in exact methods better	4.0
8. The use of a challenging mapping problem in parallel computing and tackling the problem with non-exact and metaheuristic methods is useful for guiding a parallel programming course	4.3
9. The bi-objective mapping problem is interesting enough to motivate and guide the study of parallel programming	4.5
10. Working with one challenging problem has made the work in the course more difficult than if several basic problems were stated	3.9

[8] D. J. Meder, V. Pankratius, and W. F. Tichy, http:// www.multicore-systems.org/separs/downloads/GI-WG-SurveyParallelism Curricula.pdf, 2008.

[9] R. Muresano, D. Rexachs, and E. Luque, "Learning parallel programming: a challenge for university students," in *ICCS*, no. 1, 2010, pp. 875–883.

[10] IEEE Technical Committee on Parallel Processing, Curriculum on Parallel and Distributed Computing, http://www.cs.gsu.edu/~tcpp/curriculum/index.php.

[11] F. Almeida, V. Blanco Pérez, J. Cuenca, R. Fernández-Pascual, G. García-Mateos, D. Giménez, J. Guillén, J. A. Palomino Benito, M.-E. Requena, and J. Ranilla, "An experience on the organization of the First Spanish Parallel Programming Contest," *Olympiads in Informatics*, vol. 6, pp. 133–147, 2012.

[12] Spanish Parallel Programming Contests, http://luna.inf.um.es.

[13] Z. Radivojevič, M. Cvetanović, and Z. Jovanović, "Reengineering the SLEEP Simulator in a Concurrent and Distributed Programming Course," *Comput. Appl. Eng. Educ.*, vol. 22, pp. 39–51, 2014.

[14] P. M. Oliveira, D. Vrančić, J. B. Cunha, and E. J. S. Pires, "Teaching particle swarm optimization through an open-loop system identification project," *Comput. Appl. Eng. Educ.*, vol. 22, pp. 227–237, 2014.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. The MIT Press, 1990.

[16] G. Brassard and P. Bratley, *Fundamentals of Algorithms*. Prentice-Hall, 1996.

[17] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization*. Springer, 2005.

[18] J. Hromkovič, *Algorithmics for Hard Problems*, 2nd ed. Springer, 2003.

[19] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*. New York: Wiley-Interscience, 2005.

[20] A. Cabrera, F. Almeida, V. Blanco Pérez, and D. Giménez, "Analytical modeling of the energy consumption for the high performance Linpack," in *PDP*, 2013, pp. 343–350.

[21] M. Ehrgott, *Multicriteria Optimization*. Birkhäuser, 2005.

[22] M. Mitchell, *An Introduction to Genetic Algorithm*. MIT, 1988.

[23] G. Luque and E. Alba, *Parallel Genetic Algorithms: Theory and Real World Applications*. Berlin: Springer, 2011.

[24] T. G. Crainic, M. Gendreau, and J. Y. Potvin, "Parallel tabu search," *Parallel Metaheuristics, E. Alba (ed.)*, 2005.