Using metaheuristics in a parallel computing course[⋆]

Ángel-Luis Calvo, Ana Cortés, Domingo Giménez, Carmela Pozuelo, and Miguel-Ángel Rodríguez

Departamento de Informática y Sistemas, Universidad de Murcia, Spain
angelluiscalvo@gmail.com, acc8@alu.um.es, domingo@dif.um.es, carmela@pozuelo.org, mangelrg21@gmail.com

Abstract. In this paper the use of metaheuristics techniques in a parallel computing course is explained. In the practicals of the course different metaheuristics are used in the solution of a mapping problem in which processes are assigned to processors in a heterogeneous environment, with heterogeneity in computation and in the network. The parallelization of the metaheuristics is also considered.

1 Introduction

This paper presents a teaching experience in which metaheuristic and parallel computing studies are combined. A mapping problem is proposed to the students in the practicals of a course of "Algorithms and parallel programming" [1]. The problem consists of obtaining an optimum processes to processors mapping on a heterogeneous system. The simulated systems would present heterogeneity both in the computational and network speed, and the processes to map constitute a homogeneous set, which means a HoHe (Homogeneous processes in Heterogeneous system) model is represented [2]. The mapping problem is NP [3], and it can be solved approximately through heuristics [4–6]. Each student must propose the solution of the mapping problem with some metaheuristic, so studying the characteristics of this metaheuristic and of the computation in heterogeneous environments.

The paper is organized in the following way: section 2 explains the course in which the experience has been carried out; section 3 presents the mapping problem; in section 4 the application of some of the metaheuristics considered to that problem is explained, including the parallelization of the metaheuristics; finally, section 5 summarizes the conclusions and outlines possible future studies.

2 Organization of the course

The experience has been carried out in an "Algorithms and Parallel Computing" course. This course is part of the fifth year of the studies in Computer

[⋆] This work has been funded in part by the Consejería de Educación de la Comunidad de Murcia, Fundación Séneca, project number 02973/PI/05.

Science, at the University of Murcia, in Spain. In previous years, the students have studied Algorithms and Data Structures, Computer Architecture (including multicomputers), Concurrent Programming and Artificial Intelligence. The course is optional, so the students are high level students who are interested in the subject. This, together with the fact that a reduced number of students (approximately fifteen per year) take the course, means that the teaching is personalized and focused on the work of the students. They do different studies and practicals: preparation of a presentation about some algorithmic technique, both sequential and parallel; solution and theoretical and experimental study of an algorithm to solve a challenging problem sequentially; and obtaining parallel versions (in shared memory with OpenMP and in message-passing with MPI) of the previously developed sequential algorithms. This is the first time the students work with parallel programming. The course lasts one semester and it has sequential and parallel parts, which means the parallelism is studied in approximately two months. This reduced time together with the difficulty of an initial approach to parallelims means that the goal is to introduce the students to the problems and tools of parallelism, but we do not expect them to be able to develop new algorithms and carry out detailed experiments at this stage, but they must study and program available algorithms, to adjust them to the proposed problem, to design significant experiments and to draw correct conclusions. All the information of the course (including slides of the lessons and the presentations of the students) can be found at http://dis.um.es/~domingo/app.html. In this paper we analyze the practical proposed for the first semester of the course 2006-2007.

The course centers on the solution of challenging computational problems. Thus, approximate and heuristics methods are studied, as is matricial computation, because in some of the cases such problems are matricial and numerical. Parallel programming is studied because it is used in the solution of today's highly challenging problems. Thus, the topics of the course are:

- Introduction to complexity of problems.
- Probabilistic algorithms.
- Metaheuristics.
- Matricial algorithms.
- Models of parallel programming.
- Analysis of parallel algorithms.
- Parallel algorithms.

First, the difficulties to solve some problems in a reduced time is stated. Then, some approximate, heuristics or numerical sequential algorithms are studied, and finally, the basics of parallel programming are analysed. Each student will develop sequential and parallel algorithms for the solution of a challenging problem. The proposed problem is a mapping problem where a set of identical processes is assigned to processors in a heterogeneous system. This is a NP-complete problem [4, 3], whose solution can be addressed by heuristics methods [4–6]. So, the students must tackle a challenging problem in the field of parallel programming, and they will work with topics in two parts (sequential approximate methods and parallel computing) of the syllabus. The methods proposed to solve this problem are:

- Backtracking with pruning based on heuristics (possibly pruning nodes which could lead to the optimum solution).
- Backtracking with tree traversal guided by heuristics.
- Branch and Bound with pruning based on heuristics (pruning nodes which could lead to the optimum).
- Probabilistic algorithms.
- Hill climbing.
- Tabu search.
- Scatter search.
- Genetic algorithms.
- Ant colony.
- Simulated annealing.

There are a lot of books on algorithms [7–9] and metaheuristics [10, 11] which can be consulted by the students, and also a lot of web pages to consult for more specific information.

Each student makes two presentations: one on the general ideas of the technique assigned, and the other on the parallelization with OpenMP and MPI of some algorithm which implements this technique. The presentations are previous to the practical work, so that the students can exchange ideas. The collaboration of the students is fostered. They could exchange 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, ...), and the experimental comparison of the different techniques developed by the students is positively valued in the final evaluation of the practical. In the first presentation, the technique is analysed and different ways of applying the technique to the proposed problem are considered. The students and the professor discuss alternatives. Additionally, at least two individual tutorials with each student would be organized, prior to each presentation. In the second presentation different paralelization alternatives are presented, and discusion follows.

3 The assignation problem

The problem proposed is a simplified version of a mapping problem in which the execution time of a parallel homogeneous algorithm (an algorithm in which all the processes work with the same amount of data and have the same computational cost) is used to obtain the mapping in a heterogeneous system with which the lowest possible execution time is achieved. This is a HoHe (homogeneous distribution of data and heterogeneous distribution of processes) approach, which could be preferable to a HeHo (heterogeneous distribution of data and homogeneous distribution of processes) approach because with the former it is not necessary to redesign widely tested and used parallel algorithms [12].

The method was proposed in [13], and applied to iterative parallel schemes [14] and to matrix decompositions [15]. It was explained (simplified) to the students after the study of the topics about problem complexity, probabilistic algorithms and metaheuristics, and the papers in which the method was introduced and applied, along with other related papers, were made available to students. The method is summarized below.

The execution time of a parallel algorithm is modelled as a function of some algorithmic and system parameters [16]:

$$t(s) = f(s, AP, SP) \tag{1}$$

where s represents the problem size.

The system parameters (SP) represent the characteristics of the system, and can be the cost of an arithmetic operation, or the cost of a particular arithmetic operation, the start-up time (t_s) and the word-sending time (t_w) of communications.

The algorithmic parameters (AP) are those which can be modified to obtain faster execution times. Some typical parameters in homogeneous systems are the number of processors to use from those available, or the number of rows and columns of processors in mesh algorithms. Our goal is to obtain values of the algorithmic parameters close to those which produce the lowest execution time.

The execution time model considered has the form:

$$t(s, D) = t_c t_{comp}(s, D) + t_s t_{start}(s, D) + t_w t_{word}(s, D) \tag{2}$$

where s represents the problem size, D the number of processes used in the solution of the problem, t_c the cost of a basic arithmetic operations, t_{comp} the number of basic arithmetic operations, t_s and t_w the start-up and the word-sending time, t_{start} the number of communications and t_{word} the number of data communicated.

In a homogeneous environment the values of t_c, t_s and t_w are the same in the different processors, and the the number of processes and the logical topology to use are selected to minimize the value of formula 2 (The parameters representing the topology would appear in t_{comp}, t_{start} and t_{word}.)

In a heterogeneous system it is also necessary to select the number of processes to use (D) and the number of processes assigned to each processor. These numbers are stored in an array $d = (d_1, d_2, \dots, d_P)$, with P being the number of processors in the system. The values of t_c, t_s and t_w are affected by the number of processes assigned to each processor, and are a function of d ($t_c\,(d_1, d_2, \dots, d_P)$, $t_s\,(d_1, d_2, \dots, d_P)$ and $t_w\,(d_1, d_2, \dots, d_P)$). The costs of a basic arithmetic operation in each one of the processors in the system are stored in an array t_c with P components, where t_{c_i} is the cost in processor i. And the costs of t_s and t_w between each pair of processors are stored in two arrays t_s and t_w of sizes $P \times P$, with $t_{s_{ij}}$ and $t_{w_{ij}}$ the start-up and word-sending times from processor i to processor j. These arrays may be non symmetric because communications between i and j and between j and i could have different costs. The values $t_{s_{ii}}$ and $t_{w_{ii}}$ correspond to the costs between two processes in the same processor.

The execution time model for a certain assignation of processes, d, in a system characterized by the arrays t_c, t_s and t_w would be that of equation 2, but with the values of t_c, t_s and t_w obtained from the formulae:

$$t_c = \max\{d_i t_{c_i}\}, \quad t_{s_{ij}} = \max_{d_i \neq 0, d_j \neq 0} \{t_{s_{ij}}\}, \quad t_{w_{ij}} = \max_{d_i \neq 0, d_j \neq 0} \{t_{w_{ij}}\} \tag{3}$$

In the model the cost of a basic operation in a processor is proportional to the number of processes in the processor, and no interferences are considered between processes in the same processor. Obviously, other more accurate models could be considered, but this simplified model can be used successfully in several cases [13, 15].

Obtaining an optimum mapping of processes to processors (the mapping with the lowest execution time with the model given by formulae 2 and 3) becomes a tree traversal problem if we consider the tree of all the possible mappings where each node has associated the modelled time corresponding to the mapping represented by the node. Figure 1 shows one such tree, with $P = 3$. Each level represents the possible processors to which a process can be assigned (level i corresponds to process i). There is no limit to the number of processes and the height of the tree. Because the processes are all equal, the tree is combinatorial, and because more than one process can be assigned to a processor, it includes repetitions. The tree is a logical tree which it is not necessary to store in memory, although in some methods it could be necessary to store a part. For some techniques it could be better to consider a permutational tree, for example if the technique works with a set of solutions, a permutational tree would facilitate the development of efficient routines, because it would not be necessary to include the constraints to ensure only nodes in the combinatorial tree are explored. The form of the logical tree, and the representation of the tree (or part of the tree) or the set to work with must be decided by the student.

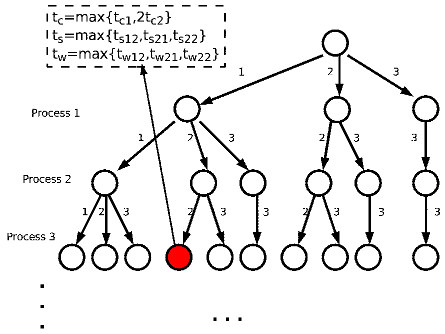
t_c=max{t_{c1},2t_{c2}}
t_s=max{t_{s12},t_{s21},t_{s22}}
t_w=max{t_{w12},t_{w21},t_{w22}}
Process 1
Process 2
Process 3
1
2
3

Fig. 1. Tree of the mappings of identical processes in a system with three processors.

Each student analyses the application to this mapping problem of an exact method with some heuristic, or of a metaheuristic technique.

First, the representation of the solutions must be decided. Each node in the solutions tree could be represented in at least two forms. In a representation with a value for each level, the grey node in figure 1 would be stated by $(1,2,2,\ldots)$, with no limit to the number of components, but a limit must be established. Because all the processes are equal, it is also possible to store the number of processes assigned to each processor. So, the grey node in the figure is represented by $(1,2,0)$.

Sequential and parallel algorithms must be developed and studied both theoretically and experimentally. The study would include the analysis of how the use of parallel computing contributes to reduce the execution time and/or the goodness of the solution. In order to have comparable results they must obtain experimental results with at least the functions:

$$t_c\frac{n^2}{5p}+t_s\frac{p(p-1)}{2}+t_w\frac{n(p-1)}{2} \tag{4}$$

which could correspond to a version of a parallel dynamic programming scheme [13], and:

$$t_c\left(\frac{2}{3}\frac{n^3}{p}+\frac{n^2}{\sqrt{p}}\right)+t_s 2n\sqrt{p}+t_w\frac{2n^2}{\sqrt{p}} \tag{5}$$

which could correspond to a version of a parallel LU decomposition [15]. Futhermore, the experiments should be carried out with the values of the system parameters in the following ranges: $1 < t_c < 5$, $4 < t_w < 40$ and $20 < t_s < 100$. Small values of t_s and t_w would simulate the behaviour of shared memory multicomputers, medium values would correspond to distributed memory systems, and large values to distributed systems.

4 Application of metaheuristics to the mapping problem

In this section the results obtained with four of the methods are shown. Three of the methods are metaheuristics methods (genetic algorithms, tabu search and simulated annealing) and the other is a backtracking with pruning based in heuristics where a node which leads to the optimum solution could be pruned. Sequential, OpenMP and MPI versions are developed in each case.

The goal is to obtain a mapping with an associated modelled time close to the optimum, but with a reduced assignation time, because this time would be added to the execution time of the routine for which the mapping is calculated. Thus, the time to optimize would be the sum of the assignation and the modelled execution times.

In the sequential algorithms the stress is put on the high algorithmic representation which allows us to obtain different versions only by changing a routine in the scheme. For metaheuristic techniques a general scheme is studied [18]. One such scheme is shown in algorithm 1.

Algorithm 1: General scheme of a metaheuristic method.

```
Inicialice($S$);
**while** *not EndCondition(S)* **do**
    $SS$ =ObtainSubset($S$);
    **if** $|SS| > 1$ **then**
        $SS1$ =Combine($SS$);
    **end**
    **else**
        $SS1 = SS$;
    **end**
    $SS2$ =Improve($SS1$);
    $S$ =IncludeSolutions($SS2$);
**end**
```

The parallel versions would allow us to obtain better mapping with the same assignation time, or mapping with similar modelled time but with a lower assignation time.

The values of the communication parameters (t_s and t_w) are obtained from two-dimensional arrays. Thus, when large systems (distributed systems) are simulated, in OpenMP the parallelization of this computation could give satisfactory results, because the system is large and, consequently, so is the computation in the loop. But in general it is normally better to parallelize at the highest possible level.

4.1 Backtracking with node pruning

Backtracking methods were used for this mapping problem in [13]. For the simulation of small systems backtracking was satisfactory, but for large systems huge assignation times were necessary. So, the work of the student was:

– For the sequential method:
  - Initially, to study backtracking methods to apply them to optimization problems. For that, typical books about algorithms were consulted [7, 8].
  - To understand the mapping problem and the increment of the assignation time when backtracking is used for large systems, which makes the backtracking impracticable in most cases.
  - To develop a backtracking scheme for the proposed mapping problem. The scheme should include a pruning routine which might be easily substitutable 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. The most representative techniques were:

  PT1 The tree is searched until a maximum level (the tree of possible mapping has not a level limit because there is no limit in the number of processes), and nodes are not pruned. This method is included as a reference which ensures the optimum mapping, supposing the number of processes is lower than the tree level.

PT2 At each step of the execution the lowest value (GLV) of the modelled execution time of the nodes generated is stored. To decide if a node is pruned a "minimum value" (NMV) is associated to it. When $NMV > GLV$ the node is pruned. In a node corresponding to p processes, NMV is obtained with a greedy method. From the execution time associated to the node new values are obtained by substituting in the model p by $p+1$, $p+2$, ... while the value decreases. NMV is taken as the minimum of of these values.

PT3 NMV is calculated in a node by substituting in the formula the value of the number of processes by the maximum speed-up achievable. For example, in node (0,2,0), with a tree like that in figure 1, the first processor will not participate in the computation, and with $t_c = (1, 2, 4)$, the relative speed-ups would be $s_r = (1, 0.5, 0.25)$, and the maximum achievable speed-up is 0.75, and this value is used instead of p.

PT4 The same value as in the previous case is used for p in the computation part, and the communication part does not vary.

- To carry out experiments to compare the results obtained with the different pruning techniques. Initially experiments were carried out for small simulated systems (between 10 and 20 processors). The best results were obtained with PT3. This technique was used in successive experiments. The main conclusion was that for small systems backtracking with pruning can be used without a large execution time and obtaining a modelled time not far from the optimum. For big systems, the mapping time is too large to be applicable in a real context. Parallelism could contribute to reduce the mapping time, so making the technique applicable.

– Different schemes were considered to obtain parallel versions, and finally a master-slave scheme was used:
  - An OpenMP version is obtained in the following way: the master generates nodes until a level; slaves are generated and all the threads do backtracking from the nodes assigned cyclically to them.
  - 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.
  - The sequential and parallel versions are compared. There is no important variation in the modelled time. The speed-up achieved when using parallelism is far from the optimum, and this is because independent backtrackings are carried out, which means less nodes are pruned with the parallel programs.

4.2 Genetic algorithm

Genetic algorithms are possibly the most popular metaheuristic techniques. The students saw this technique on a previous course on Artificial Inteligence. The work of the student was:

- For the sequential method:
  - To consult some of the books about genetic algorithms [19] and the numerous stuff in the web.
  - To understand the mapping problem and to identify population and individual representations to apply genetic algorithms to the problem.
  - To identify how the routines in algorithm 1 could be for the genetic scheme.
  - To develop a genetic scheme at a high level. The scheme must allow to easily change some parameters (i.e. number of individual in the pobla-tion, number of iterations to converge, ...) or routines (i.e. mutation, combination, ...).
  - To experimentally tune the values of the parameters and the routines to apply (modifing them with in the high level scheme developed) to the mapping problem. The principal conclusion was that to obtain a reduced assignation time (and so a version applicable to the mapping problem) it is necessary to reduce the number of individuals and the number of iterations to achieve the convergence, but on the other hand this reduction would produce a reduction in the goodness of the solution. Satisfactory results (both for the assignation time and the goodness of solution) are obtained from experiments with 10 individuals and with convergence after 10 iterations with no improvement. In any case, genetic algorithms do not seem to be the most adequate metaheuristica technique for this problem, because normally the size of the population is large and that produce a large assignation time.
- About the parallel versions:
  - The OpenMP program works by simply parallelizing the combination of the population.
  - From the different parallel genetic schemes [20], the island scheme was selected for the message-passing version. The number of generations to exchange information between the islands is one parameter to be tuned.
  - The sequential and parallel versions are compared. With OpenMP the same mappings are found, but with an important reduction in the assignation time. In MPI this time is not reduced substantially, but better mappings are normally obtained.

4.3 Tabu search

While genetic algorithms make a global search, tabu search is a local search technique which uses memory structures to guide the search. The students saw this technique on a previous course on Artificial Intelligence. The work of the student was:

- For the sequential method:
  - To consult some of the books about metaheuristics methods [10, 11] and the numerous stuff in the web.

  - To understand the mapping problem and to identify set and element representations to apply tabu search to the problem.
  - To identify how the routines in algorithm 1 would be for tabu search.
  - To develop a tabu search at a high level. 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, ...) or routines (i.e. generation of the first stage, ...).
  - To experimentally tune the values of the parameters and the routines to apply (modifying them in the high level scheme) to the mapping problem. Satisfactory results are obtained when: the number of iterations a movement is tabu is equal to half the number of simulated processors (P); the initial stage is obtained by assigning P processes to the fastest processors; the number of iterations to begin the diversification phase is three quarters of the maximum number of iterations, ...

– About the parallel versions:

  - The OpenMP program works by selecting at each step a number of nodes to explore equal to the number of available processors.
  - For the MPI version, different tabu techniques have been studied [17]. A pC/RS/MPDS technique has been used: each process controls its own search; knowledge is not shared by the processes; multiple initial solutions are used; and different search strategies are used. To diversify the search, some processes start with heuristic solutions and others with ramdom solutions, and the number of iterations a movement is tabu depends on the number of the process.
  - The sequential and parallel versions are compared. In OpenMP the speedup is satisfactory, and in some cases superlinear. In MPI the time is not reduced substantially, and only a small improvement in the mappings is obtained. A small reduction in the execution time can be achieved by reducing the number of iterations in the MPI version.

4.4 Simulated annealing

Simulated annealing is also a local search technique. The students saw this technique in a previous course on Artificial Intelligence. The work of the student was:

– For the sequential method:

  - To consult some of the books about metaheuristics methods [10, 11] and the numerous stuff in the web.
  - To understand the mapping problem and to identify set and element representations to apply simulated annealing to the problem.
  - To identify how the routines in algorithm 1 would be for simulated annealing.

  - To develop a simulated annealing at a high level. The scheme must allow to easily change some parameters (i.e. initial temperature, cardinality of the neighborhood set, ...) or routines (i.e. generation of the initial node, neighborhood, cooling function, ...).
  - To experimentally tune the value of the parameters and the routines to apply (modifying them in the high level scheme developed) to the mapping problem. Satisfactory results are obtained when the cardinality of the neighborhood set is set to 10. For different execution time models satisfactory mappings and with a reduced execution time are obtained for different values of the initial temperature and different cooling functions. So, no conclusions are obtained at this point.
- About the parallel versions:
  - The OpenMP program works by parallelizing the computation of the actual values of the system parameters (t_c, t_s and t_c).
  - As in the tabu search, because the search is local, to parallelize at a high level different search are carried out in the different processes. The best solution in each process is communicated to the other processes periodicaly, and each process continues with the best solution at this moment.
  - The sequential and parallel versions are compared. In OpenMP the parallelism is exploited at a low level, and a reduction of the mapping time is achieved for large problem sizes. In MPI the reduction in the mapping time is small, and the modelled time is a little better than in the sequential case.

5 Conclusions and possible future studies

The paper presents a teaching experience using metaheuristics in combination with parallel computing in a course of “Algorithms and Parallel Programming”. With this combination the students work at the same time with two of the topics of the course, the importance of approximate methods and heuristics is better understood when working with a challenging problem like the one proposed, and the difficulty and importance of the mapping problem is better understood working in the problem with a metaheuristic approach. Futhermore, parallel programming is introduced using the same metaheuristics with which the mapping problem is tackled, and the parallelism at different levels and in shared memory and message-passing is considered. In addition, because all the students work with the same mapping problem, but each student works in a different mapping algorithm, collaboration between the students and the common enrichment is fostered.

This first experience has been very positive, and so it will be continued in successive courses. At the moment other mapping problems in the field of parallel computing are being considered: the mapping of a tasks graph to a graph representing a hierarchical cluster; the mapping of a loop of tasks with different costs to a system with computational and memory heterogeneity, among others.

References

1. Web page of the Algorithms and Parallel Computing course at the University of Murcia, http://dis.um.es/~domingo/app.html
2. A. Kalinov and A. Lastovetsky, Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Network of Heterogeneous Computers, *Journal of Parallel and Distributed Computing*, 61 (4), 2001, pp 520-535.
3. H. Lennerstad and L. Lundberg, Optimal Scheduling Results for Parallel Computing, *SIAM News*, 1994, pp 16-18.
4. W. Zhao and K. Ramamritham, Simple and Integrated Heuristic Algorithms for Scheduling Tasks with Time and Resource Constrains, *The Journal of Systems and Software*, 7 (3), 1987, pp 195-205.
5. S. Fujita, M. Masukawa and S. Tagashira, A Fast Branch-and-Bound Scheme for the Multiprocessor Scheduling Problem with Communication Time, in: *Proceedings of the 2003 International Conference on Parallel Processing Workshop (ICPPW'03)*, Kaohsiung, Taiwan, 2003, pp 104-111.
6. G. Sabin, R. Kettimuthu, A. Rajan and P. Sadayappan, Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment, in: D. G. Feitelson, L. Rudolph and U. Schwiegelshohn, eds., *Job Scheduling Strategies for Parallel Processing, 9th International Workshop*, Seattle, 2003, Lecture Notes in Computer Science, vol. 2862, pp 87-104.
7. G. Brassard and P. Bratley. *Fundamentals of Algorithms*, Prentice-Hall, 1996.
8. T. H. Cormen, C. E. Leiserson and R. L. Rivest *Introduction to Algorithms*, MIT Press, 1991.
9. D. Giménez, J. Cercera, G. García y N. Marín, *Algoritmos y Estructuras de Datos, Vol II, Algoritmos*, Diego Marín, 2003.
10. J. Dréo, A. Pétrowski, P. Siarry and E. Taillard. *Metaheuristics for Hard Optimization*, Springer, 2005.
11. Juraj Hromkovič. *Algorithmics for Hard Problems*, Second Edition, Springer, 2003.
12. A. Kalinov and S. Klimov, "Optimal mapping of a parallel application processes onto heterogeneous platform", *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, Colorado, USA, 2005.
13. J. Cuenca, D. Giménez and J. P. Martínez-Gallar, Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems, *Parallel Computing* **31** (2005) 771-735.
14. J. P. Martínez, F. Almeida and D. Giménez, *Mapping in heterogeneous systems with heuristical methods*, Workshop on state-of-the-art in Sci. Par. Comp., Umeå, Sweden, June 18-21, 2006.
15. J. Cuenca, L. P. García, D. Giménez and J. Dongarra, *Processes Distribution of Homogeneous Parallel Linear Algebra Routines on Heterogeneous Clusters*, in Proc. IEEE Int. Conf. on Cluster Computing, IEEE, HeteroPar05, 27-30 September 2005, Boston, Massachusetts.
16. J. Cuenca, D. Giménez and J. González, *Architecture of an Automatic Tuned Linear Algebra Library*, Parallel Computing, 30 (2), 2004, pp 187-220.
17. T. G. Crainic, M. Gendreau and J. Y. Potvin, *Parallel Tabu Search*, in Parallel Metaheuristics, E. Alba (ed.), John Wiley & Sons, 2005.
18. G. R. Raidl, *A unified view on hybrid metaheuristics*, in Hybrid Metaheusistics, Third International Workshop, Gran Canaria, October 2006, pp 1-12.
19. Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, Springer-Verlag, 1992.

20. G. Luque, E. Alba, B. Dorronsoro, *Parallel Genetic Algorithms*, in E. Alba (ed.) Parallel Metaheuristics. A New Class of Algorithms, chapter 5, Wiley, 2005.