Application of metaheuristics to tasks-to-processors assignation problems in heterogeneous systems

Francisco Almeida[1], Javier Cuenca[2], Domingo Giménez[3], Juan Pedro Martínez Gallar[4]

Abstract— **The optimization of the execution time of a parallel algorithm can be achieved through the use of an analytical cost model function representing the running time. Typically the cost function includes a set of parameters that models the behaviour of the system and the algorithm. In order to reach an optimal execution, some of these parameters must be fitted according to the input problem and to the target architecture. An optimization problem can be stated in which the modelled execution time for the algorithm is used to estimate the parameters. Due to the large number of variable parameters in the model, analytical minimization techniques are discarded. Exhaustive search techniques can be used to solve the optimization problem, but when the number of parameters or the size of the computational system increases, the method is impracticable. The use of metaheuristics allows the development of valid approaches to solve general problems with a large number of parameters. The combination of the parameterized analytical cost model function and metaheuristic minimization methods appears as a real alternative to minimize the parallel execution time in complex systems. The success of the approach is shown with two different algorithmic schemes on parallel heterogeneous systems.**

Keywords— **Processes mapping, Metaheuristics, Heterogeneous computing.**

I. Introduction

THE minimization of the running time of a parallel program on a parallel architecture is a challenging problem in current target platforms. Parameters like block sizes and shapes, number and set of processors, process to processor assignment, etc. must be fitted for a proper optimal execution time. Common methodologies to achieve this vary from ad-hoc profiling and tuning of the concrete program on the specific architecture, or solving scheduling problems associated to the dependence task graphs, to the minimization of the analytical complexity cost function associated to the program. Each of the methodologies has advantages and disadvantages and their use is sometimes a matter of availability of tools and know-how.

We focus on minimization strategies through the cost model function. The complexity needed in the model to be accurate for minimization purposes is too high. However, when the model is simple enough to be managed, the predictive abilities of the model allow us not only to minimize the execution time but also to provide the opportunity to use them as automatic tuning engines. Figure 1 illustrates the steps sequence of the method. The analytical cost function of the algorithm is developed in the design phase. The inclusion of some parameters in the function allows the behaviour of the platform and the algorithm to be modelled. The values which model the behaviour of the platform are estimated and included in the cost function when the routine is installed. Variations in the values of some of the algorithmic parameters affect the speed of the program. The optimal combination of these parameters is obtained at running time, and the routine is executed with values which provide the lowest execution time according to the cost function.

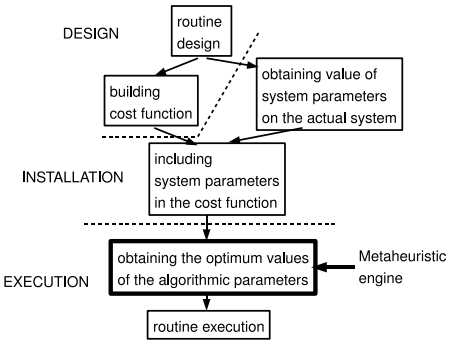

Fig. 1. Block diagram of the steps to obtain an execution close to the optimum in execution time.

The minimization problem increases in difficulty when the size or complexity of the system increases. Analytical minimization has been successfully applied in some specific cases but it is discarded in practice for medium and large sized systems. Exhaustive algorithmic search techniques can be used, but when the number of parameters or the size of the computational system increases, the method is impracticable due to time restrictions. The use of approximated methods to guide the search is another choice, although approximated methods are typically ad-hoc methods and need the use of information about the objective function, with the consequent loss of generality. Furthermore, these methods very frequently are trapped in local minimums. We propose metaheuristics as a part of the general methodology. This

[1]Departamento de Estadística, I. O. y Computación, Universidad de La Laguna, e-mail: falmeida@ull.es.
[2]Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: javiercm@ditec.um.es.
[3]Departamento de Informática y Sistemas, Universidad de Murcia, e-mail: domingo@dif.um.es.
[4]Departamento de Estadística, Matemáticas e Informática, Universidad Miguel Hernández, Alicante, e-mail: jp.martinez@uhm.es.

approach has been successfully applied in several specific systems [1], which opens the way for the generalization proposed here.

Due to space limit, this paper summarizes the proposal, but a deeper study and more experiments can be found in the references [2], [3], [4]. Section two summarizes the concepts used in the development of parameterized models of the execution time. In section three the basic ideas of metaheuristic methods are explained. Section four shows the application of the proposed methodology to two schemes. Finally, section five presents some conclusions.

II. Parameterized model of the execution time

We propose to express the execution time of a parallel algorithm as a function of some algorithmic and system parameters as follows [5]:

$$t(s) = f(s, AP, SP) \quad (1)$$

where s represents the problem size, SP are the parameters (system parameters) which represent the characteristics of the system, and AP are the parameters (algorithmic parameters) whose modification gives different versions of the algorithm and which can be modified to obtain low execution times. The number of system parameters may be very large, and the formula could become very complex. Some typical algorithmic parameters are the number of processors to use from those available in a homogeneous system, the number of rows and columns of processors in logical mesh algorithms, the block computational size... In a heterogeneous system the number of parameters increases greatly, because different block sizes could be preferable in different processors, because a different number of processes can be assigned to each processor... So, the optimization problem for heterogeneous or distributed environments is particularly complex.

Let nap be the number of algorithmic parameters, and vsp the values of the system parameters. For each algorithmic parameter (AP_i, $1 \leq i \leq nap$) the possible values of the parameters form a set (SAP_i, $1 \leq i \leq nap$) which could be very large (we call $nvap_i = |SAP_i|$). Figure 2 shows a tree to obtain the possible combinations of the algorithmic parameters. Each terminal node represents a combination, and there is a total of $\prod_{i=1}^{nap} |SAP_i|$ terminal nodes. The optimization problem to be solved is:

$$t_{opt} = min_{vap \in SAP_1 \times \ldots \times SAP_{nap}} \{f(s, vap, vsp)\} \quad (2)$$

We propose the systematic application of metaheuristic methods to approach the optimum solution of tasks-to-processors assignation problems in which the minimum modelled time from those of all the possible assignations is searched.

III. Metaheuristics

Given that most of the interesting and attractive combinatorial problems belong to the NP class, exact methods are not very useful except for small sized problems. In recent decades, metaheuristics have emerged as an advantageous technology for approximation algorithms [1]. All the existing metaheuristics share ideas and differ in some components. The unified views of metaheuristics presented, for example, in [6], provide an interesting general vision of the field. A general algorithm like that shown in figure 3 could be instantiated to obtain different metaheuristics just by changing the key elements (`Initialize`, `EndCondition`, `ObtainSubset`, `Combine`, `Improve`, `IncludeSolutions`) and the sizes of the involved sets of solutions. This conception allows for the development of frameworks or toolboxes where operators could be reused and combined in specific metaheuristics. For example, when a scatter search method is generated, the generation of a genetic algorithm may reuse several routines from the scatter search, and some of the new procedures implemented on the genetic technique can be used for the scatter search in a feedback process.

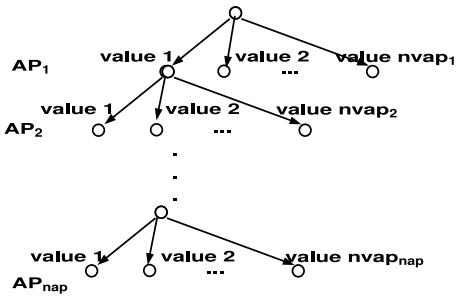

Fig. 2. A tree to explore all the possible assignations to the algorithmic parameters.

```
Initialize(S)
while (not EndCondition(S)) {
    SS = ObtainSubset(S)
    if(|SS| > 1)
        SS1 = Combine(SS)
    else
        SS1 = SS
    SS2 = Improve(SS1)
    S = IncludeSolutions(SS2)
}
```

Fig. 3. General scheme of a metaheuristic method.

IV. Case studies

Two cases are studied to validate the general methodology. They cover quite complex situations and the analytical objective functions to be optimized are very different.

A. Processes to processors allocation for iterative schemes through scatter search

Two main approaches have been traditionally followed to adapt parallel codes efficiently in heterogeneous systems. The first approach considers the heterogeneous data distribution between the processes.

A second technique consists of performing a proper allocation of processes to processors, and to use homogeneous algorithms. This type of algorithms is called HeHo [7] (Heterogeneous assignment of processes which have homogeneous distribution of data). Many works have been devoted to the first approach [8], [9], which typically would lead to efficient algorithms but at a high cost of reprogramming classical algorithms. For the second approach to be successful, it is necessary to obtain a good distribution strategy of the processes. The algorithmic parameters include a vector of parameters, $d = (d_1, \ldots, d_D)$, where d_i holds the processor assignment of process i, being D the number of processes. The values of the arithmetic costs vary from one processor to another, and also according to the number of processes assigned to each processor. The values of the communication parameters also vary between each pair of processes, depending on the processors to which the processes are assigned. The cost of a basic computational operation in each processor i is t_{c_i}, and the start-up and word-sending times from processor i to processor j are $t_{s_{i,j}}$ and $t_{w_{i,j}}$. Equation 1 takes the form:

$$t(s, D) = t_c t_{comp}(s, D) + t_{comm}(s, D) \qquad (3)$$

where s represents the problem size, t_{comp} the number of arithmetic basic operations, and t_{comm} the communication cost (it includes a term with t_s and another with t_w). t_{comp} and t_{comm} are divided in different parts, and in each part the value of the process with most computational charge, and that of the pair of processes with most communication cost, are considered. The values of t_c, t_s and t_w are affected in some way by the number of processes assigned to each processor. The way in which these values vary with the processes assignation also depends on the characteristics of the computational system and the communication network. One initial approach is to consider the cost of a basic arithmetic operation in processor i as $d_i t_{c_i}$, and that the communication costs have the maximum value of those between processors where some process is assigned.

The cost function depends on the routine in question, and is obtained in the design phase. An iterative scheme is used to exemplify the methodology. Figure 4 shows the basic algorithm where every process has an amount of work assigned to compute in each iteration and after each iteration a communication operation is performed.

```
// proc = Process ID
// Problem size at each node
count[proc] = Problem_Size / Num_procs
for (i = 0; i < N; i++) {
// Each node computes the work of size
    count
   Work (count)
// Each node sends its job to all nodes
   Communication_operation ()
}
```

Fig. 4. Basic algorithm of an iterative scheme.

We instantiate our technique for dynamic programming problems as particular cases of iterative schemes. In a basic dynamic programming algorithm the solution is obtained by completing a bidimensional table where the columns represent the size of the corresponding subproblem (states), and the rows are the number of maximum decisions (stages). The final solution is in the last row and column. The optimal solution with i stages and problem size j is obtained from optimal solutions with fewer than i previously stored decisions. These dependencies mean that a parallel version of the scheme will have a number of steps equal to the `number of stages`. In a message passing version of the scheme, because different data are assigned to different processors a communication step appears between two consecutive computation steps.

For instantiation purposes, we will consider dynamic programing approach for the "coins problem". Given the quantity C and coins of n types $(v_1, v_2, \ldots, v_n)$, with a certain amount of coins of each type $(q_1, q_2, \ldots, q_n)$, the *coins problem* consists of finding the minimum number of coins to fulfil amount C. As part of the methodology, we follow a sequence of steps to achieve the optimal process to processor allocation (figure 1). These steps are discussed following.

Step 1: Identification of parameters

The problem size is determined by C and by the number of different values of coins. The values of the coins and the amount of coins available of each type also affect the execution time, and hence they are also part of the problem size, $s = (n, C, v, q)$.

The system parameters are different for each processor or pair of processors. If there are P processors in the system, t_c is represented by a vector $t_c = (t_{c_1}, \ldots, t_{c_P})$, and t_s and t_w by matrices of size $P \times P$.

The number of processes to use (D) is one of the algorithmic parameters, and another is the vector d representing the processes to processors distribution, so, $AP = (D, d)$.

Step 2: Routine modelling

An analysis of the computation/communication pattern composes the analytical model. The total amount of computations of an iteration are assigned to the D processes. The C states in a step are assigned to the D processes and the optimal value for state X in a given stage i, is given by the formula:

$$M(i, X) = \min_{k=0,1,\ldots,min\left\{\left\lfloor \frac{X}{v_i} \right\rfloor, q_i\right\}} \{M(i-1, X - kv_i) + k\} \qquad (4)$$

If the columns of the table are assigned using a block distribution, $\frac{C}{D}$ consecutive columns are assigned to each process. The process with most work is process number D, and the work assigned in step i to this process is:

$$\sum_{j=C-\frac{C}{D}+1}^{C} \left(1 + min\left\{\left\lfloor \frac{j}{v_i} \right\rfloor, q_i\right\}\right) \quad (5)$$

Each process i, with $i = 1, 2, \ldots, D-1$ sends the values it has computed to the processes $i+1, i+2, \ldots, D$. The amount of transferred data is $\frac{C(D-1)}{2}$, and the number of communications $\frac{D(D-1)}{2}$. In an Ethernet network the cost of communications can be modelled as:

$$t_{comm} = \frac{D(D-1)}{2} t_s + \frac{C(D-1)}{2} t_w \quad (6)$$

Step 3: Obtaining the values of the system parameters

The values of the system parameters can be obtained by running a reduced version of the routine or the section of the routine where the parameters appear.

The values of t_s and t_w could be obtained by using a ping-pong between each pair of processors. But a ping-pong is not the type of routine which appears in the communication phase in the algorithm, and bad predictions will be made. Another possibility is to estimate these values by running a small version of the problem in parallel for different values of the algorithmic parameters.

The values of t_c can be obtained by running a small sequential version of the code in each processor.

Step 4: Selection of the values of the algorithmic parameters

An optimization problem consisting of deciding the values of D and d which produces the lowest theoretical execution time needs to be solved. Different metaheuristic techniques can be used [4]. We use a scatter search method to show how to find the AP parameters.

We consider that each element of the reference set represents a possible mapping of processes to processors. In scatter search the elements with the lowest modelled execution times are included in the reference set, and those with the greatest distance function to the best elements are also included. The reason is that if only the best elements of the reference set are selected, the method could quickly converge to a local optimum. The structure to represent the reference set is the mapping array d, and $d_i = j$ when process i is allocated to processor j.

One specific scheme of the scatter search can be obtained from the general metaheuristic scheme of figure 3. There are different possibilities for each one of the routines in the scheme. Some of them are discussed below:

- `Initialize`: The initial generation may consider several options, varying from a random assignment of processes to processors, where processors with more computational capacity receive more probability, to the allocation of an equal number of processes to each processor. The elements are improved to procure good solutions from the beginning. A greedy method is applied to each element of the initial reference set and of the sets obtained by combination of elements. For each element, all the mappings obtained by assigning one additional process to a processor are considered. The modelled execution times of these mappings are computed and the method moves to the mapping with lowest time. The iteration finishes when the execution times of all the new mappings are greater than that of the previous one.
- `EndCondition`: We can consider that the convergence is reached when the best new solutions are not better than the best of the previous ones, or when the average of the new solutions is not better than that of the previous ones.
- `ObtainSubset`: It is possible to select all the elements for combination, or to select the best elements to be combined with the worst ones.
- `Combine`: Each pair of selected elements is combined component to component, and the number of processes increases with more probability in processors with more computational capacity, and decreases in the slower ones.
- `Improve`: The same greedy method used in the initialization is used to improve each generated element.
- `IncludeSolutions`: The most promising elements and those most scattered with respect to the most promising ones are included in the reference set. The most scattered elements are the most distant elements (using Euclidean distance) or those which are most "different" (number of different coordinates) from the most promising ones.

Step 5: Routine execution

The routine is executed with the best selection of algorithmic parameters. Table I summarizes the behaviour of the metaheuristic when compared with a backtracking with node pruning with simulations named AE-LD, AE-MD, BW-LD and BW-MD. The table shows the percentage of times the mapping time plus the modelled time with the scatter search is lower than that of the backtracking. The metaheuristic method is a better approach than the backtracking, even when some node elimination which could eliminate the optimum solution is used.

TABLE I
Comparison of two selection and inclusion criteria with respect to backtracking with node pruning.

	Select elem.	
Include elem.	All elements	Best vs worst elem.
Longest dist.	AE-LD: 90%	BW-LD: 85%
More dif.	AE-MD: 91%	BW-MD: 86%

Table II compares the 4 previous combinations from representative simulations. BW-LD and BW-MD show the lower decision time however, the Total Time (Final Modelled Time, FMT, plus Decision Time, DT) is lower in AE-LD and AE-MD. AE-MD has a number of iterations smaller than AE-LD, but

the Total Time are similar in both strategies.

TABLE II
Modelled times and number of iterations for different criteria.

Simul.	Iter.	FMT	DT
AE-LD	96	1796.04	5.89
AE-MD	69	1797.83	5.33
BW-LD	11	2237.24	0.18
BW-MD	2	2411.21	0.47

We perform a deeper analysis in more complex systems using the AE-MD for the Scatter Search. We include a parameter, the complexity, that reflects the complexity of computations in comparison with communications. Table III shows the grouped results for simulations with growing sizes and complexities. For each combination of size, complexity, number of processors and number of processes, 20 executions were made. Each percentage represents the number of simulations where the Scatter Search technique is better than backtracking with pruning.

TABLE III
Comparison of backtracking with pruning and Scatter Search techniques with increasing sizes and complexities

Sim:		1	2	3	4	5
Size	Compl.	20 p	40 p	60 p	80 p	100 p
100000	100	100%	100%	60%	43%	33%
100000	400	100%	100%	96%	96%	70%
750000	100	100%	100%	96%	94%	77%
750000	400	100%	98%	100%	98%	97%

B. Assignation of tasks with a master-slave scheme

In this case we are interested in the use of metaheuristics as a cooperative framework to minimize the analytical function. Nevertheless, the work in the five steps of the methodology is explained here.

Step 1: Identification of parameters

A master processor generates a set of T independent tasks that are solved by slave processors. The task have arithmetic costs $c = (c_1, c_2, \ldots, c_T)$ and memory requirements $r = (r_1, r_2, \ldots, r_T)$. The system consists of P processors with the times to perform a basic arithmetic operation $t_c = (t_{c_1}, t_{c_2}, \ldots, t_{c_P})$, and memory capacities $m = (m_1, m_2, \ldots, m_P)$. The problem size is $s = (T, c, r, P, m)$ and the only system parameter is t_c, because no communications are considered. From all the mappings of tasks to the processors, $d = (d_1, d_2, \ldots, d_T)$, with $r_i \leq m_{d_i}$, the problem is to find d, the algorithmic parameter, that minimizes the modelled execution time.

Step 2: Routine modelling

The execution time for a mapping d is:

$$\max_{i=1,\ldots,P} \left\{ t_{c_i} \sum_{j=1 \ / \ d_j=i}^{T} c_j \right\} \qquad (7)$$

The minimum of the times of the mappings which satisfy the memory constraints must be obtained.

Step 3: Obtaining the values of the system parameters

As in the previous case the values of t_c can be obtained by running a small sequential version of the code in each processor, or the relative speed of the processors can be used.

Step 4: Selection of the values of the algorithmic parameters

The tasks are generated by the master at running time, so the mapping must be decided in a reduced time. There is a maximum of P^T assignations, and it is not possible to solve the problem by generating all the possible mappings. An alternative is to obtain an approximate solution using some metaheuristic method. The use of a generic scheme allows us to develop different metaheuristics easily and to experiment with them to decide a satisfactory method at installation time or to experiment with different methods to obtain a satisfactory solution at running time.

Step 5: Routine execution

The routine is executed with the best selection of algorithmic parameter values. The use of a metaheuristic method (that determined in the installation) or of several methods to decide the mapping at running time adds an overhead time which must be low, and some time criteria could be included in the stopping condition.

B.1 Application of different metaheuristics

The use of a generic scheme allows us to develop different metaheuristics easily and to experiment with them. Experiments with different metaheuristic are shown. The methods are: genetic algorithm (GA), scatter search (SS), tabu search (TS) and GRASP (GR). These metaheuristics are analysed from the same perspective, identifying common routines and element representations, which allows us to reuse some routines, so reducing the development time. They can be implemented using the same class structure: classes for the algorithm, the system and the solution. Each metaheuristic is also implemented in a class, but they share the pattern and some functions.

The selection of the best functions and parameters' values is obtained by experimentation. Experiments with different tasks and systems configurations have been carried out, obtaining similar results. The results of a particular experiment are shown. The size (n) of each task has been randomly generated between 1000 and 2000, the arithmetic cost is n^3, and the memory requirement n^2. The number of processors in the system is the same as the number of tasks. The costs of basic arithmetic operations has been randomly generated between 0.1 and 0.2 $\mu secs$. The memory of each processor is between half the memory needed by the biggest task and one and a half times this memory. Experiments have been carried out for each metaheuristic with different functions

and with values of the basic parameters with which a mapping time lower than 3 seconds is obtained for the biggest problem considered. Satisfactory results have been obtained in the following conditions:

- GA. The population has 80 elements; the elements in S are generated randomly assigning the tasks to the processors, with the probability proportional to the processor speed. Each pair of elements is combined with half of the components of each parent; in each combination the best parent and the best descendant are included in the population. The probability of mutation is 1/5. The maximum number of iterations is 800, and the maximum number of iterations without improving the optimum is 80.
- SS. S has 20 elements. The initialization and combination are those in GA. Each element is improved with a greedy method, which works by selecting from the processor with highest execution time a task that, assigned to another processor, could reduce the fitness function. The elements with lowest cost function and those most scattered with respect to the best ones are included in the reference set. The maximum number of iterations is 400, and the maximum number of iterations without improving the optimum is 40.
- TS. The neighborhood has 10 elements, obtained by taking the tasks assigned to the processor with most cost and reassigning them. The tabu list has 10 elements. The maximum number of iterations is 200, and the maximum number of iterations without improving the solution is 20.
- GR. An initial set with 20 elements is generated, as in GA and SS. The element is selected randomly, with more probability for the elements with better fitness function. The element is improved with the greedy method used in SS. The number of iterations is 20.

Table IV shows the mapping and the simulated times. GA and SS are the methods that need more mapping time. GR is the method which behaves best. There are no big differences in the mapping times, but the differences in the modelled times are big in some cases. Of course, all the methods must be carefully tuned to the problem to obtain the best combination of parameters and functions.

TABLE IV
Comparison of the metaheuristics. Mapping time and modelled execution time (in seconds), varying the number of tasks.

	GA	SS	TS	GR
tasks	mapping time			
50	0.213	0.159	0.004	0.006
100	0.198	0.337	0.006	0.016
200	0.205	0.582	0.061	0.045
400	1.111	1.257	0.156	0.153
800	1.831	2.613	0.328	0.372
tasks	modelled execution time			
50	1566	1900	1757	**1524**
100	1903	1961	3018	**1460**
200	**3452**	**3452**	**3452**	**3452**
400	**3069**	3910	**3069**	**3069**
800	3276	3121	2746	**1571**

V. Conclusion and future works

We state the advantages of using metaheuristics to approach the minimization of the modelled execution time of parallel programs. An optimization problem consisting of obtaining the parameters with the lowest modelled execution time is stated. The general problem is NP, and the use of heuristics approaches is compulsory. A general framework for the application of metaheuristics techniques to this problem is presented. With this approach several metaheuristics solutions can be easily developed, studied and combined. The approach has been illustrated with two case studies. We plan to apply this methodology to other more complex mapping problems. The development of a set of classes containing different implementations of the general metaheuristic scheme is being considered. With such a set of classes, a user could implement the mapping problem to be solved, and easily develop several metaheuristics to experiment with the problem. The approach could be used for the solution of similar mapping problems in which the function to optimize has a different form.

Acknowledgment

This work has been funded in part by Fundación Séneca, project 02973/PI/05, and by the EC (FEDER) and the Spanish MEC with the I+D+I contract TIN2005-09037-C02-01.

Referencias

[1] F. Glover and G. A. Kochenberger, *Handbook of Metaheuristics*, Kluwer, 2003.

[2] 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*, vol. 31, pp. 717–735, 2005.

[3] J. Cuenca, D. Giménez, J. J. López-Espín, and J. P. Martínez-Gallar, "A proposal of metaheuristics to schedule independent tasks in heterogeneous memory-constrained systems," in *Proc. IEEE Int. Conf. on Cluster Computing*. September 2007, IEEE Computer Society.

[4] D. Giménez, A.-L. Calvo, A. Cortés, and C. Pozuelo, "Using metaheuristics in a parallel computing course," in *ICCS08, Proceedings International Conference on Computational Science, Vol II, LNCS*. 2008, vol. 5102, Springer.

[5] J. Cuenca, D. Giménez, and J. González, "Architecture of an automatic tuned linear algebra library," *Parallel Computing*, vol. 30, no. 2, pp. 187–220, 2004.

[6] G. R. Raidl, "A unified view on hybrid metaheuristics," in *Hybrid Metaheuristics, Third International Workshop, LNCS*, October 2006, vol. 4030, pp. 1–12.

[7] 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*, vol. 61, no. 44, pp. 520–535, 2001.

[8] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A proposal for heterogeneous cluster scalapack (dense linear solver)," *IEEE Transactions on Computers*, vol. 50, no. 10, pp. 1052–1070, 2001.

[9] C. Banino, O. Beaumont, A. Legrand, and Y. Robert, "Sheduling strategies for master-slave tasking on heterogeneous processor grids," Tech. Rep. 2002-12, INRIA, 2002.