Improving Metaheuristics for Mapping Independent Tasks into Heterogeneous Memory-Constrained Systems^{*}

Javier Cuenca¹ and Domingo Giménez²

¹ Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia,
30071 Murcia, Spain

javiercm@ditec.um.es

² Departamento de Informática y Sistemas, Universidad de Murcia, 30071 Murcia,
Spain

domingo@dif.um.es

Abstract. This paper shows different strategies for improving some metaheuristics for the solution of a task mapping problem. Independent tasks with different computational costs and memory requirements are scheduled in a heterogeneous system with computational heterogeneity and memory constraints. The tuned methods proposed in this work could be used for optimizing realistic systems, such as scheduling independent processes onto a processors farm.

Keywords: processes mapping, metaheuristics, heterogeneous systems.

1 Introduction

In this work the problem of mapping independent tasks to the processors in a heterogeneous system is considered. The tasks are generated by a processor and sent to other processors which solve them and return the solutions to the initial one. So, a master-slave scheme is used. The master-slave scheme is one of the most popular parallel algorithmic schemes [1], [2]. There are publications about optimal mapping master-slave schemes in parallel systems [3], [4], [5], but in those works the optimal mappings are obtained only under certain restrictions, and memory constraints are not considered.

In our approach each task has a computational cost and a memory requirement. The processors in the system have different speeds and a certain amount of memory, which imposes a restriction on the tasks which it can be assigned. The goal is to obtain a task mapping which leads to a low total execution time. To obtain the optimum mapping in the general case is an NP problem [6], and heuristic methods may be preferable. In our previous work [7], the basic scheduling problem was explained together with some possible variants. To solve them, different metaheuristics (Genetic Algorithm, Scatter Search, Tabu Search and

^{*} This work has been partially supported by the Consejería de Educación de la Región de Murcia, Fundación Séneca 02973/PI/05.

GRASP) [8], [9] were proposed. In this work these metaheuristics are improved in different ways in order to reduce the time to perform the task mapping and to obtain a better solution.

The paper is organized in the following way: in section 2 the basic scheduling problem is explained; in section 3 some metaheuristics for the solution of the proposed scheduling problem are analysed; and, finally, section 4 summarizes the conclusions and outlines future research.

2 Scheduling Problem

Of the different scheduling problems introduced in our previous work [7], this paper studies, as an example, the problem with fixed arithmetic costs and no communications in depth. In this problem, given t tasks, with arithmetic costs $c = (c_0, c_1, \dots, c_{t-1})$ and memory requirements $i = (i_0, i_1, \dots, i_{t-1})$, and p processors with the times to perform a basic arithmetic operation $a = (a_0, a_1, \dots, a_{p-1})$, and memory capacities $m = (m_0, m_1, \dots, m_{p-1})$, from all the mappings of tasks to the processors, $d = (d_0, d_1, \dots, d_{t-1})$ ($d_k = j$ means task k is assigned to processor j), with $i_k \leq m_{d_k}$, find d with which the following minimum is obtained:

$$\min_{\{d/ i_k \leq m_{d_k} \forall k=0,1,\dots,t-1\}} \max_{\{j=0,1,\dots,p-1\}} \left\{ a_j \sum_{l=0,1,\dots,t-1; d_l=j} c_l \right\}, \quad (1)$$

where the minimum of the mapping times which satisfies the memory constraints is obtained, and for each mapping the time is that of the processor which takes most time in the solution of the tasks it has been assigned.

There is a maximum of p^t assignments (with the memory constraints the number of possibilities may decrease), and it is not possible to solve the problem with a reasonable time by generating all the possible mappings. An alternative is to obtain an approximate solution using some heuristic method. This possibility is considered in this paper.

3 Application of Metaheuristics to the Scheduling Problem

In this section the application of metaheuristic methods to the version of the scheduling problem previously described is analysed. The methods considered are: Genetic Algorithm (GA), Scatter Search (SS), Tabu Search (TS) and GRASP (GR). The four metaheuristics are analysed from the same perspective, identifying common routines and element representations. The goal is to obtain a mapping with an associated modelled time close to the optimum, but with a low assignment time, because this time is added to the execution time of the routine. A general metaheuristic scheme is considered [10]. One such scheme is shown in algorithm 1. Each of the functions that appears in that scheme works in a different way depending on the metaheuristic chosen:

Algorithm 1. General scheme of a metaheuristic method.

```

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

```

- **Initialize.** To create each individual of the initial set S , this function assigns tasks to processors with a probability proportional to the processor speed.
 - **GA** works with a large initial population of assignments.
 - **SS** works with a reduced number of elements in S . This could produce a lower time for this method than that of the **GA**.
 - **TS** works with a set S with only one element.
 - **GR:** In each iteration the cost of each candidate is evaluated, and a number of candidates are selected to be included in the set of solutions.
- **ObtainSubset:** In this function some of the individuals are selected randomly.
 - **GA:** The individuals with better fitness function (equation 1) have more likelihood of being selected.
 - **SS:** It is possible to select all the elements for combination, or to select the best elements (those with better fitness function) to be combined with the worst ones.
 - **TS:** This function is not necessary because $|S| = 1$.
 - **GR:** One element from the set of solutions is selected to constitute the set SS ($|SS| = 1$).
- **Combine:** In this function the selected individuals are crossed, and $SS1$ is obtained.
 - **GA, SS:** The individuals can be crossed in different ways. One possibility is to cross pairs of individuals by exchanging half of the mappings, obtaining two descendants.
 - **TS, GR:** This function is not necessary.
- **Improve:**
 - **GA:** A few individuals are selected to obtain other individuals, which can differ greatly. This process is done by using mutation operands. The aim is to diversify the population to avoid falling in local optimums.
 - **SS:** This function consists on a greedy method which works by evaluating the fitness value of the elements obtained with the p possible processors (with memory constraints) in each component, in order to search for a better element in its neighborhood.

- **TS:** Some elements in the neighborhood of the actual element are analysed, excluding those in a list of previously analysed tabu elements.
- **GR:** This function consists of a local search to improve the element selected. Some greedy method can be used, or all the elements in the neighborhood of the selected one can be analysed.
- **IncludeSolutions:** This function selects some elements of $SS2$ to be included in S for the next iteration.
 - **GA:** The best individuals from the original set, their descendants and the individuals obtained by mutation, are included in the next S .
 - **SS:** The best elements are selected, as well as some elements which are scattered with respect to them to avoid falling within local minimums.
 - **TS, GR:** The best element from those analysed is taken as the next solution.
- **EndCondition:**
 - **GA, SS, TS, GR:** The convergence criterion could be a maximum number of iterations, or that the best fitness value from the individuals in the population does not change over a number of iterations.

3.1 Basic Experimental Tuning of the Metaheuristics

Experiments with different tasks and systems configurations have been carried out, obtaining similar results. The experiments, whose results are shown beyond, has the following configuration: The size 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 μsecs . The memory of each processor is between half the memory needed by the biggest task and one and a half times this memory. Preliminary results for the proposed problem in section 2 have been obtained using the following parameter values, whereas with other close values the results would be similar.

- **GA:**
 - **Initialize:** The population has 80 elements; the elements in S are initially generated randomly assigning the tasks to the processors, with the probability proportional to the processor speed.
 - **Combine:** 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.
 - **Improve:** the probability of mutation is $1/5$.
 - **EndCondition:** the maximum number of iterations is 800, and the maximum number of iterations without improving the optimum solution is 80.
- **SS:**
 - **Initialize:** S has 20 elements. The initialization is that in **GA**.

- **Combine:** The combination is that in **GA**.
 - **Improve:** Each element is improved with a greedy method, which works by selecting for the processor with highest execution time a task which could be assigned to another processor reducing the fitness function (equation 1).
 - **IncludeSolutions:** The elements with lowest cost function and those most scattered with respect to the best ones (using a 1-norm) are included in the reference set.
 - **EndCondition:** The maximum number of iterations is 400, and the maximum number of iterations without improving the optimum solution is 40.
- **TS:**
- **Improve:** The neighborhood has 10 elements, obtained by taking the tasks assigned to the processor with most cost and reassigning them to other processors. The tabu list has 10 elements.
 - **EndCondition:** The maximum number of iterations is 200, and the maximum number of iterations without improving the solution is 20.
- **GR:**
- **Initialize:** The initial set has 20 elements. The elements are generated as in **GA** and **SS**.
 - **ObtainSubset:** The element selected from S is chosen randomly, with more probability for the elements with better fitness function (equation 1).
 - **Improve:** The element is improved with the greedy method used in **SS**.
 - **EndCondition:** The number of iterations is 20.

Table 1 compares the mapping time and the simulated time obtained, in a PC Centrino 2.0 GHz., with each of the heuristics, and those with a backtracking, for those problem sizes where the backtracking obtains a solution using a reasonable mapping time. Those cases where the corresponding method does not obtain the optimal solution are in bold. In almost all the cases the metaheuristics provide the best solution and use less time than a backtracking.

Table 1. Comparison of backtracking and the metaheuristics. Mapping time and modelled execution time (in seconds), varying the number of tasks.

tasks	Back		GA		SS		TS		GR	
	map.	simul.	map.	simul.	map.	simul.	map.	simul.	map.	simul.
4	0.025	3132	0.051	3132	0.065	3132	0.010	3132	0.019	3132
8	0.034	4731	0.028	4731	0.132	4731	0.015	4731	0.024	4731
12	0.058	1923	0.021	1923	0.158	1923	0.016	2256	0.029	1923
13	0.132	1278	0.055	1278	0.159	1278	0.016	1376	0.024	1278
14	0.791	1124	0.081	1124	0.192	1124	0.017	1124	0.027	1135

For big systems and using the different heuristics, satisfactory mappings are obtained in a reduced time. In Table 2 the mapping and the simulated times for big systems are shown. Those cases where the best solution of modelled time is obtained for each problem size appear in bold. **GA** and **SS** are the methods that need more mapping time to obtain a good solution with the parameters considered. **GR** and **TS** use much less time and obtain the best solution for almost all the cases. **TS** needs less time than **GR**, but its solutions are not always as good. Therefore, **GR** is the method which behaves best.

Following these results with the preliminary tunings, a deeper study on how to improve those metaheuristics is now underway. For example, the next subsection shows how advanced tunings can be applied to the Genetic Algorithm.

Table 2. Comparison of the metaheuristics for big systems. Mapping time and modelled execution time (in seconds), varying the number of tasks.

tasks	GA		SS		TS		GR	
	map.	simul.	map.	simul.	map.	simul.	map.	simul.
25	0.139	1484	0.259	1450	0.010	1450	0.045	1450
50	0.413	1566	0.429	1900	0.015	1757	0.078	1524
100	0.592	1903	0.834	1961	0.022	3018	0.158	1460
200	0.825	3452	1.540	3452	0.079	3452	0.293	3452
400	3.203	3069	2.682	3910	0.375	3069	0.698	3069

3.2 Advanced Tuning of the Genetic Algorithm

Various tunings possibilities have been studied in order to improve the **GA** method. The most significant are:

- In the routine **Combine**:
 - **T1.** It is possible to change the heredity method. Instead of a descendant inheriting strictly each half of its components from each parent, each component is inherited pseudo-randomly, so giving more probability to the parent with best fitness value (the fitness value of a solution is the modelled execution time of the processor that needs more time to finish its assigned tasks) (equation 1).
 - **T2.** Another possibility of changing the heredity method consists of choosing each component of a descendant from the less loaded processor from those of its parents. The load of a processor r , W_r , is the product of the cost of performing an arithmetic operation in r and the addition of the cost of the tasks assigned to r :

$$W_r = a_r \sum_{\{l=0,1,\dots,t-1;d_l=r\}} c_l, \quad (2)$$

In other words, if for the i -th component, that task is assigned in the parent A to the processor r , that has a load of W_r , and in the parent B to the processor q , that has a load of W_q , then in the descendant the component i will be r if $W_r < W_q$, or q in other case.

- **T3.** In the routine **Improve** it is possible to introduce a hybrid approach, using a steered mutation instead of a pure mutation. In the solution to be improved, each task assigned to an overloaded processor (a processor is overloaded if its load (equation 2) is greater to the average load of all the processors) is reassigned randomly to another processor. Therefore, this routine mutates the solution to another where the total loads of the most overload processors have been reduced.
- **T4.** In the routine **ObtainSubset**, where the solutions that will be combined are chosen, it is possible to chose these solutions pseudo-randomly, giving more probability to the solutions with better fitness.

In the first column of Table 3 the times obtained with the base case (the original **GA**) in a PC Centrino 2.8 Ghz, are shown for different numbers of tasks. In the second column, the times obtained when the **T1** tuning are shown. The solutions are obtained more quickly than in the base case (less mapping time), but these solutions are worse than the previous ones (more simulated execution time). This could be because **T1** is a greedy tuning that leads the algorithm to a local minimum. In the third column, the times obtained when the **T2** tuning are shown. Now, the time to obtain the solutions is very similar to the base case, but the solutions for some of the problems are better. So this tuning could be an interesting improvement. In the forth column the times obtained when the **T3** tuning is applied to the routine **Improve** are shown. The solutions are worse than in the base case, that is, a steered mutation does not work as well as we thought (a deeper study is given below). In the fifth column, the times obtained when the **T4** tuning is applied to the routine **Improve** are shown. The times are better in some cases. It converges faster than the base case, using less mapping time. Since the improvements **T2**, **T3** and **T4** seem interesting and they affect different parts of the algorithm, it could be appropriate to combine them. In this way in the sixth and seventh columns of Table 3 the results when using **T2** with the other tunings are shown. Combining **T2** with **T3** or **T4** the results do not improve those obtained only with **T2** and they need more mapping time. Therefore, it is better to apply just **T2**. Finally, combining **T3** and **T4** the results do not improve any of them.

Table 3. Comparison of the different tunings applied to the Genetic Algorithm, varying the number of tasks

tasks	basic GA		T1		T2		T3		T4		T2+T3		T2+T4		T3+T4	
	map.	simul.	map.	simul.	map.	simul.	map.	simul.	map.	simul.	map.	simul.	map.	simul.	map.	simul.
50	0.13	1646	0.02	2277	0.05	1524	0.08	1715	0.09	1715	0.05	1524	0.06	1524	0.08	1715
100	0.25	2068	0.09	2581	0.13	1460	0.14	2230	0.25	2000	0.17	1460	0.16	1460	0.14	2230
150	0.47	2422	0.19	2908	0.19	2039	0.25	2464	0.36	2418	0.22	2039	0.22	2039	0.25	2464
200	0.41	3452	0.28	3717	0.31	3452	0.31	3452	0.33	3452	0.34	3452	0.34	3452	0.33	3452
400	1.56	3069	1.19	4184	1.19	3069	1.67	3069	1.42	3069	1.20	3069	1.25	3069	1.72	3069
1600	12.10	3680	10.50	4061	11.77	1735	11.38	3882	12.08	3482	12.56	1735	11.28	1735	12.09	3882

In order to understand better the behavior of the algorithm with the different tunings, the Figs. 1, 2 and 3 show the evolution of the best solution from the new generated individuals per iteration, in each case, along all the iterations, for the problem of mapping 1600 tasks.

Fig. 1. Evolution of the best solution from the new generated individuals per iteration for a problem size of 1600 tasks. Without tuning (T0) applied to the routine **Combine**, with T1 and with T2.

Regarding the routine **Combine** (Fig. 1), if the tuning **T1** is applied, the restriction of inheriting each component from the best parent confers a more greedy tendency to the algorithm. It falls in local minimums, with worse solutions than in the base case, where it can seldom exit. However, with the tuning **T2** each component of a descendant can come from any of the parents, so a bigger

Fig. 2. Evolution of the best solution from the new generated individuals per iteration for a problem size of 1600 tasks. Without tuning (T0) applied to the routine **Improve** and with T3.

Fig. 3. Evolution of the best solution from the new generated individuals per iteration for a problem size of 1600 tasks. Without tuning (T0) applied to the routine **ObtainSubset** and with T4.

mixture of genetic code is produced, causing more diversity of descendants and so allowing the algorithm exits from local minimums easily. The tendency, from the first iteration, is to improve the best solution because the most overloaded processors are unloaded in each step.

In the routine **Improve** (Fig. 2), with the tuning **T3** the mutation operation is steered towards better solutions quickly, but this kind of mutation prevents the genetic code of the descendant differing a lot from those of the parents. In this way, if the algorithm falls in a local minimum it is very difficult to get out of it because it has not a pure mutation.

If the tuning **T4** is applied to the routine **ObtainSubset** (Fig. 3), the algorithm progresses slowly but surely, because in each iteration only the best solutions are chosen to have descendants and few false moves are made.

4 Conclusions and Future Works

The paper presents some improvements on previous proposals for the application of metaheuristics techniques to tasks to processors mapping problems, where the tasks are independent and have various computational costs and memory requirements, and the computational system is heterogeneous in computation and with different memory capacities (communications are not yet considered).

The metaheuristics considered have been: Genetic Algorithm, which is a global search method; Scatter Search is also a global search method, but with improvement phases; Tabu Search is a local search method with the search guided by historic information; GRASP method is a multiple local search method. The

parameters and the routines have been tuned and the experiments to obtain satisfactory versions of the metaheuristics have been carried out, mainly with the Genetics Algorithm where some detailed tuning techniques have been studied.

In future works advanced tunings, like those applied to the Genetic Algorithm in this work, will be applied to the other metaheuristics. On the other hand, different characteristics of the heterogeneous systems will be considered: variable arithmetic cost in each processor depending on the problem size, variable communication cost in each link,... Other general approximations (dynamic assignation of tasks, adaptive metaheuristics,...) will also be studied. The tuned methods proposed in this work will be used for optimizing realistic systems, such as scheduling independent processes or mapping MPI jobs onto a processors farm.

References

1. Wilkinson, B., Allen, M.: *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, 2nd edn. Prentice-Hall, Englewood Cliffs (2005)
2. Grama, A., Gupta, A., Karypis, G., Kumar, V.: *Introduction to Parallel Computing*, 2nd edn. Addison-Wesley, Reading (2003)
3. Banino, C., Beaumont, O., Legrand, A., Robert, Y.: Scheduling strategies for master-slave tasking on heterogeneous processor grids. In: Fagerholm, J., Haataja, J., Järvinen, J., Lyly, M., Råback, P., Savolainen, V. (eds.) *PARA 2002*. LNCS, vol. 2367, pp. 423–432. Springer, Heidelberg (2002)
4. Pinau, J.F., Robert, Y., Vivien, F.: Off-line and on-line scheduling on heterogeneous master-slave platforms. In: *14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2006)*, pp. 439–446 (2006)
5. Brucker, P.: *Scheduling Algorithms*, 1st edn. Springer, Heidelberg (2007)
6. Lennerstad, H., Lundberg, L.: Optimal scheduling results for parallel computing. *SIAM News*, 16–18 (1994)
7. Cuenca, J., Giménez, D., López, J.J., Martínez-Gallary, J.P.: A proposal of metaheuristics to schedule independent tasks in heterogeneous memory-constrained systems. In: *CLUSTER* (2007)
8. Hromkovic, J.: *Algorithmics for Hard Problems*, 2nd edn. Springer, Heidelberg (2003)
9. Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: *Metaheuristics for Hard Optimization*. Springer, Heidelberg (2005)
10. Raidl, G.R.: A unified view on hybrid metaheuristics. In: Almeida, F., Blesa Aguilera, M.J., Blum, C., Moreno Vega, J.M., Pérez Pérez, M., Roli, A., Sampels, M. (eds.) *HM 2006*. LNCS, vol. 4030, pp. 1–12. Springer, Heidelberg (2006)