Contributions to the Multiprocessor Scheduling Problem

Bernard Chauvière, Dominique Geniet
L.I.S.I.
Université de Poitiers & ENSMA
F-86961 Futuroscope
bernard.chauviere@ensma.fr
dominique.geniet@univ-poitiers.fr

René Schott
IECN and LORIA
Université Henri Poincaré-Nancy 1
F-54506 Vandœuvre-lès-Nancy
schott@loria.fr

Abstract. The problem of multiprocessor scheduling consists in finding a schedule for a general task graph to be executed on a multiprocessor system so that the schedule length can be minimized. This scheduling problem is known to be NP-hard, and methods based on heuristic search have been proposed to obtain optimal and suboptimal solutions. Efficient methods based on genetic algorithms have been developed by (just to name a few) Hou, Ansari, Ren, Wu, Yu, Jin, Schiavone, Corrêa, Ferreira, Reybrend,..., to solve the multiprocessor scheduling problem. In this paper, we propose various algorithmic improvements for the multiprocessor scheduling problem. Simulation results show that our methods produce solutions closer to optimality than [3] when the number of processors and/or the number of precedence constraints increase.

1 Introduction

Multiprocessor scheduling has been a source of challenging problems for researchers in the area of computer engineering. The general problem of multiprocessor scheduling can be stated as scheduling a set of partially ordered computational tasks onto a multiprocessor system so that a set of performance criteria will be optimized. Since this scheduling problem is known to be NP-hard, various heuristic approaches have been developed to solve the problem [1, 3, 4, 5]. In this paper, starting from the genetic algorithm designed by Hou, Ansari and Ren [3], we present several improvements for the genetic approach, then we switch to a method based on the gradient and to several variations of tabu search. This paper is organized as follows. First we present briefly in Section 2 the model for multiprocessor scheduling and some preliminaries of genetic algorithms. In Section 4 we explain our contributions. Section 5 contains simulation results and performance comparisons with the genetic approach of [3]. Section 6 presents some further research aspects and concluding remarks.

2 Model and definitions

We use the graph model proposed in [3].
A set of partially ordered computational tasks is represented by a directed acyclic graph consisting of finite nonempty set of vertices, V and a set of finite directed edges, E, connecting the vertices. The collection of vertices, $V = T_1, T_2, \ldots, T_m$, represents the set of computational tasks to be executed. Let e_{ij} denote a directed edge from vertex T_i to T_j. The set of the directed edges, $E = e_{ij}$ implies a partial ordering or precedence relation, $>>$, exists between the tasks. $T_i >> T_j$ means that task T_i must be completed before T_j can be initiated. The problem of optimal scheduling a task graph onto a multiprocessor system with p processors is to assign the computational tasks to the processors so that the precedence relations are maintained and all of the tasks are completed in shortest possible time. If task T_i is an ancestor of task T_j (i.e., if T_i must be executed before T_j), then we say that $height(T_i) < height(T_j)$ where $height(T_i) = 0$ if the set $PRED(T_i)$ of predecessors of T_i is empty and $height(T_i) = 1 + max_{T_j \in PRED(T_j)}(T_j)$ otherwise. This height function conveys the precedence relations between tasks.
[2] is an excellent introduction to genetic algorithms. We just recall the principles. Typically, a genetic algorithm consists of

the following steps.
1) Initialization-an initial population of the search nodes is randomly generated.
2) Evaluation of the fitness function-the fitness value of each node is calculated according to the fitness function.
3) Genetic operations (crossover, mutation)-new search nodes are generated randomly by examining the fitness value of the search nodes and applying the genetic operators to the search nodes.
4) Repeat steps 2 and 3 until the algorithm converges.

3 Contributions

In this section we outline the modifications/improvements of the genetic algorithm (GAI) proposed in [3], then we present a method based on the gradient and several algorithmic variations around the tabu search.

3.1 Improvements of GAI

[3] proposes a genetic algorithm algorithm for solving the multiprocessor scheduling problem. As mentioned in [1], this method fas some drawbacks. In this section we present two improvements which allow to overcome the difficulties of [3].

The method proposed by [3] for generating the initial population, attributes incrementaly the tasks to the processors. Let $T(h)$ be the set of tasks of height h and max_h the depht of the precedence graph. For each height h, a random number n_h of tasks is attributed to the first processor. n_h is chosen uniformly in $[0, h]$. On the average, the number of tasks attributed to the first processor is given by:

$$\frac{n_1}{2} + \frac{n_2}{2} + \ldots + \frac{n_{max_h}}{2} = \frac{n}{2}$$

This process is repeated for the following processors with the set of non-attributed tasks. The set of remaining tasks is attributed to the last processor. The average number of tasks attributed to the k^{th} processor is given by $n/2^k$. Therefore, the tasks are not uniformly distributed on the processors. This fact leads to bad performances (see Figure 4) when the number of processors increases. In order to improve tha quality of the initial population, we propose to repartition schemes:

1. Uniform repartition of the number of tasks attributed to each processor: we upperbound the number of tasks attributed to a processor by $\lceil n/p \rceil$.

2. Uniform load balancing: we upperbound the load of each processor by $\lceil \frac{\sum_{k=1}^{n} C_k}{p} \rceil$.

In that follows GAV (resp. GAX) stands for our genetic algorithm based on the first (resp. second) proposed repartion schema. These modifications improve the quality of the obtained solutions when the number of processors increases. There is nevertheless room for further improvements and a deeper investigation has to be done. [1] uses directly the precedence graph instead of the height of the tasks, in order to design new genetic operators but their approach increases the time complexity.

3.2 A method based on the gradient

In this section we present a gradient decrease method in order to approximate the multiprocessor scheduling problem. We use again the model of [3] for representing the solutions. The cost function is also defined asin [3]: it corresponds to the time necessary for executing all tasks with respect to the indicated order.

The method called "decrease of the gradient" consist in successive modifications of a solution. For so doing, we evaluate the neighbor solutions and keep only the best. When the algorithm fails in improving a solution it corresponds to a local minimum and the algorithm returns that result. One of the key points of this gradient method relates to the definition of the neighborhood of a solution: it has to be as small as possible (because of the execution time) but has to allow the exploration of the largest possible part of the space of solutions in order to guarantee a good quality for the solution.

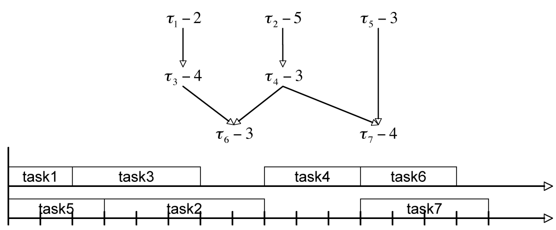

Figure 1: An example of scheduling

Consider the execution sequence indicated in Figure 1. We transform it into a neighbor solution thanks to two types of modifications:

1. the permutation of two adjacent tasks attributed to the same processor (see Figure 2),

2. the change of the processor attributed to a task (see Figure 3).

In that follows, DGI stands for the algorithm based on the gradient method applied to the neighborhood as defined above.

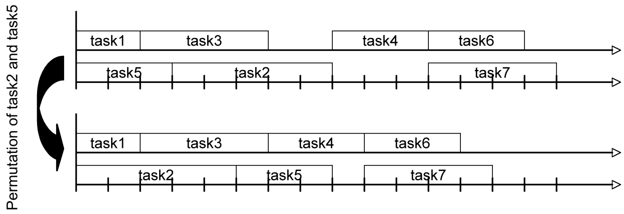

Figure 2: Permutation of two adjacent tasks

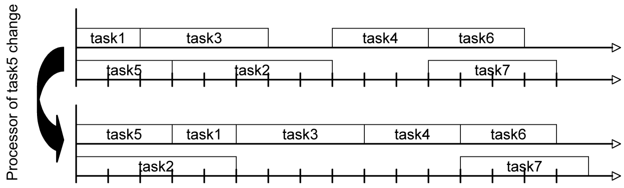

Figure 3: Change of the processor attributed to a task

3.3 Variations on Tabu Search

Tabu Search is a gradient method with a local minima avoiding mechanism. Some solutions or characteristics are forbidden. For example, if a local solution is a local minimum, then there must be a possibility for reducing the quality of the current solution and also to forbid the return to this local minimum for some time.

Our first implementation (called DGV) of Tabu Search consist in forbiding definitively the already reached local minima. The second implementation (called DGX) forbids some characteristics of the solutions. In Section 3.2, we have defined the neighborhood of a solution with the help of two types of modifications. To each of them we are going to associate a specific tabu characteristic:

1. for a processor attributed to k tasks, there exist $k-1$ possible permutations between adjacent tasks, we identify them in a natural way by their position. When the algorithm escapes from a local minimum thanks to that modification, the corresponding permutation becomes tabu.
2. when the algorithm escapes from a local minimum by changing the processor attributed to a task, the previous attribution becomes tabu: this task cannot be attributed again to the previous processor.

Forbidding definitively some characteristics is not intersting. In the next section, we determine the number of forbidden iterations.

3.4 Complexity of the neighborhood of a solution

The two modifications which we can apply to a solution define the neighborhood of that solution. In this section, we count the number of neighbors of a solution. For so doing we assume *that each processor is attributed to at least one task*. We use the following notations:

- n: the number of tasks,
- p: the number of processors,
- $\sigma(i)$: the number of tasks attributed to the i^{th} processor.

For the i^{th} processor, there are $\sigma(i)-1$ possible task permutations, as soon as each processor is attributed to at least one task. For the set of all processors we get therefore V_1 neighbors, as soon as all tasks are attributed exactly to a processor:

$$V_1 = \sum_{k=1}^{p}(\sigma(i)-1) = (\sum_{k=1}^{p}\sigma(i)) - p = n - p$$

Consider now the modification consisting in changing the processor attributed to a task. A given task j attributed to the processor i can be attributed to $p-1$ other processors. Assume that the task is attributed to the k^{th} processor. There are $\sigma(k)+1$ possible insertion positions. For the $p-1$ processors, we get V'_j possible neighbors:

$$V'_j = \sum_{k=1,k\neq j}^{p}(\sigma(k)+1) = n+p-1-\sigma(i)$$

For the set of tasks associated to the i^{th} processor, we get $V_i^{\star}$ neighbors:

$$V_i^{\star} = \sigma(i)(n+p-1-\sigma(i))$$

The second modification of a solution generates therefore V_2 neighbors:

$$\begin{aligned} V_2 &= \textstyle\sum_{k=1}^{p}\sigma(k)(n+p-1-\sigma(k)) \\ &= (n+p-1)\textstyle\sum_{k=1}^{p}\sigma(k) - \sum_{k=1}^{p}\sigma(k)^2 \\ &= n(n+p-1) - \textstyle\sum_{k=1}^{p}\sigma(k)^2 \end{aligned}$$

Finally, as soon as a processor is attributed to at least one task and that all tasks are attributed to at least one processor, the number V of neighbors is given by:

$$V = V_1 + V_2 = n(n+p) - p - \sum_{k=1}^{p}\sigma(k)^2$$

The above expression depends on the current solution. But $\sum_{k=1}^{p}\sigma(k)^2 \leq n^2$ because $\sum_{k=1}^{p}\sigma(k) = n$. Therefore:

$$(n-1)p \leq V \leq n(n+p) - p$$

We must fix the number of forbidden iterations of a tabu characteristic. Our experimentations have shown that this value depends indeed on the number of tasks and on the number of processors. The above bounds led us to take np for that value.

4 Simulation results

In order to compare the different scheduling methods, we have generated randomly systems of 50 tasks. Three tests have been performed for respectively 25, 50 and 100 precedence relations. For each test, the number of processors varies. Each sample contains 100 task systems.

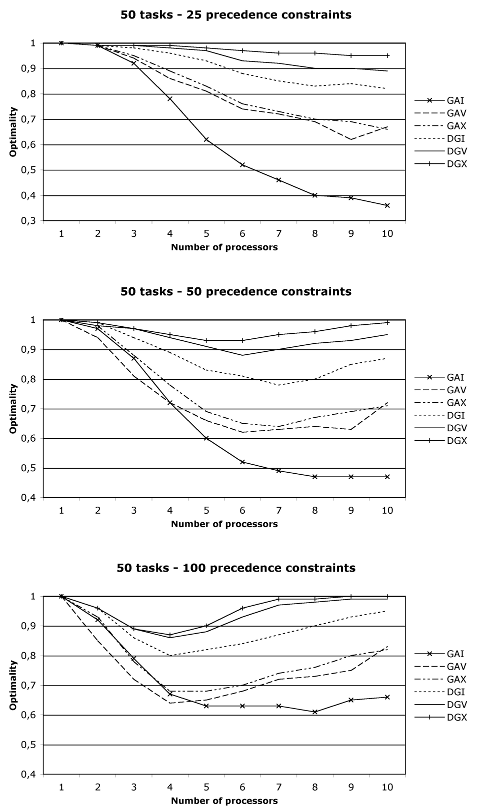

Figure 4: Performances of each scheduling method

The indicator used in order to evaluate the quality of the solutions produced by a method, is the ratio between the minimal theoretical time necessary for executing all tasks of a system and the execution time of the produced solution. We consider two lowerbounds of the execution time of a task system:

- the first lowerbound corresponds to a full functioning all available processors (precedence relations are not taken into account):

$$\lfloor \frac{\sum_{k=1}^{n} C_k}{p} \rfloor$$

- the second lowerbound corresponds to the execution time of longest precedence chain.

The minimal theoretical time is defined as the maximum of the two above values. Figure 4 describes the quality of the solutions produced by each method with respect to the minimal execution time. Each point of the curves corresponds to the average quality of the solution obtained for each of the 100 task systems of the sample. When the number of processor and/or the number of precedence constraints increase, it appears that: GAV and GAX outperform GAI, DGI produces even better solutions than GAV and GAX, DGV and DGX produce solutions very close to the minimal theoretical execution time

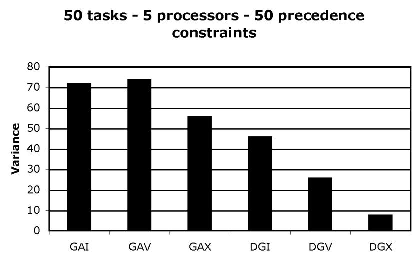

Figure 5: Variance of each scheduling method

Figure 5 shows the variance of each method (100 executions of each method on a system with 50 tasks, 50 precedence relations, on a 5 processors machine). DGX and DGV have the smallest variances.
We are cuurently implementing the genetic algorithm of [1]. Comparison results will be presented in an upcoming version of this extended abstract.

5 Conclusions

The various methods presented in this paper provide important performance improvements of multiprocessor scheduling when the number of processors as well as the number of precedence constraints increase. There is still room for further algorithmic improvements. Designing more advanced hybrid algorithms for multiprocessor scheduling seems promising but difficult. A method based on Markov Decision Processes is in progress by the authors. Using similar methods for task scheduling on real-time mutiprocessor systems is also an important research topic.

References

[1] Corrêa, R.C., Ferreira, A., and Rebreyend, P, Scheduling Multiprocessor Tasks with Genetic Algorithms, IEEE Trans. Parallel and Distributed Systems, 10, 8, 825-837, 1999.

[2] Goldberg, D.E., Genetic Algorithms in Search, Oprtimization, and Machine Learning, Addison-Wesley, 1989.

[3] Hou, E.S., Ansari, N., and Ren, H., A Genetic Algorithm for Multiprocessor Scheduling, IEEE Trans. Parallel and Distributed Systems, 5, 2, 113-120, 1994.

[4] Monnier, Y., Beauvais, J-P., and Déplanche, A.M., A Genetic Algorithm for Scheduling Tasks in a Real-Time Ditributed System, Proceedings of the 24th EUROMICRO Conference (EUROMICRO'98), IEEE Pub.

[5] Wu, A.S., Yu, H., Jin, S., Lin, K-C, , and Sciavone, G., An Incremental Genetic Algorithmic Approach to Multiprocessor Scheduling, IEEE Trans. Parallel and Distributed Systems, 15, 9, 824-834, 2004.