7th IFAC Conference on Manufacturing Modelling, Management, and Control
International Federation of Automatic Control
June 19-21, 2013. Saint Petersburg, Russia

Optimization Of Matching and Scheduling On Heterogeneous CPU/FPGA Architectures

OMAR SOUISSI [*] RABIE BEN ATITALLAH [*] DAVID DUVIVIER [*] ABDELHAKIM ARTIBA [*]

[*] *LAMIH UMR 8201, University of Valenciennes and Hainaut-Cambrésis, Le Mont Houy, 59313 Valenciennes cedex 9 - France*

Abstract:
The continuous improvements in several application domains such as telecommunications, aerospace or multimedia lead to additional design constraints on power budget and architecture scalability. The heterogeneous CPU/FPGA (Central Processing Unit/Field-Programmable Gate Array) architecture is one of the most promising solutions in this context leading to high performance reconfigurable computing. In such systems, multi-core processors (CPU) provide high computation rates while the reconfigurable logic (FPGA) offers high performance and adaptability to the application real-time constraints. However, there is a lack of CAD (Computer Aided Design) tools able to deal with the development of applications on such heterogeneous systems. This research investigates the problem of the optimisation of static and run-time task mapping on a real-time computing system CPU/FPGA used to implement intimately coupled hardware and software models.

Keywords: Algorithm, Heuristic, Heterogeneous computing, Static and Run-Time mapping.

1. INTRODUCTION

A promising approach to cover the needs of future systems with real time constraints are heterogeneous architectures with a combination of general CPUs (Central Processing Unit) and reconfigurable fabrics provided by standard FPGAs circuit (Field-Programmable Gate Array). In such systems, the multi-core CPUs provide raw computation rates and ease of programming while the reconfigurable logic offers massively parallel computing power and adaptability on the circuit level. Due to the high parallelism rate of the application, FPGA technology could offer better performances comparing to the CPUs (up to ten times Asano et al. (2009) at lower frequencies). FPGA is a chip containing a large number of logic blocks, these blocks are connected together by a configurable routing matrix, which allows the reconfiguration of the component functionality as desired.

In order to harvest the maximum benefits of the performance of each part of the system (CPU and FPGA), we must offer efficient methods and tools that help software designers to map efficiently the application tasks on the available resources while considering real-time constraints. This challenge involves the static and the runtime task mapping. As highlighted in Jong-Kook et al. (2007), an important research challenge is how to assign tasks to the available resources in order to maximize some performance criterion of the heterogeneous architecture.

In this research, we explore the problem of task mapping in terms of minimizing the makespan [1]. We assume that the tasks execution is non-preemptive, and we take into account the communications cost between each pair of tasks linked by precedence. We formulate an exact method and a heuristic approach in order to deal, respectively, with static initial mapping and runtime mapping in case of the arrival of new tasks. The effort to develop an exact method was motivated by its importance in the static initial phase, especially in view of the absence of contributions in the literature concerning exact approaches. The heuristic that we introduced is a *Modified* version of the well known HEFT (*Heterogeneous Earliest-Finish Time*) Heuristic. Indeed, experiments in many prior contributions in the literature have shown that the HEFT Heuristic is one of the most effective heuristics to deal with heterogeneous architectures.

This paper is structured as follows. Section 2 reviews some representative related contributions. Sections 3 describes the problem statement and the context of this study. The exact method and the MHEFT Heuristic are presented in detail in Sections 4 and 5; respectively.

2. RELATED WORKS

Several works in the field of task mapping in computing systems have been proposed, such as Koulamas and Kyparisis (2007), Chen and Lin (1998) and ulrich Heiss (1992), which target identical parallel machines. Recently, research efforts are increasingly oriented towards solving the mapping problem onto heterogeneous computing systems. The mapping problem consists of two significant parts; the *matching* that involves assigning tasks to the available resources, and the *scheduling* that considers the execution sequence of tasks. In the literature, the majority of the existing works refer to the term *scheduling* to mean *mapping*. For example, Ibarra and Kim (1977) presents several heuristic algorithms for scheduling independent tasks on non-identical processors. Already in 1988, Casavant and Kuhl (1988) presented a taxonomy of scheduling in general-purpose distributed computing systems. At the highest level, they distinguish between *local scheduling*, which we called scheduling, and *global scheduling*, which we called matching as shown in Figure 1. At the lower level they differentiate

[1] Informally speaking, the makespan is the time difference between the start and finish times of a set of jobs or tasks.

between two categories of mapping: static and dynamic; see Figure 1. A number of papers propose different approaches to address the problem of static mapping, see Shoukat et al. (2002) and Braun et al. (2001), in which a comparison of eleven static heuristics is presented for mapping a class of independent tasks onto heterogeneous distributed computing systems. Additionally several papers have investigated the dynamic mapping problem; see Jong-Kook et al. (2007), Fazlali et al. (2010) and Lin et al. (1997). The different approaches explored include the greedy heuristics in Luo et al. (2007), in which the authors evaluate and compare 20 greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. In Alaoui et al. (1999), the authors offer a genetic algorithm for the mapping problem; finally a hybrid heuristic is presented in Chen and Lin (1998). To summarize, most of the approaches in the literature can be classified into two categories mainly, heuristic-based and guided-random-search based algorithms.

The first category, namely, heuristic-based algorithms, may be classified into three groups:

- **list scheduling**, such as the well-known heuristics HEFT (*Heterogeneous Earliest-Finish Time*) and CPOP (*Critical-Path-On-a-Processor*) presented in Topcuoglu and Wu (2002) ;
- **cluster scheduling**, such as the DSC (*Dominant Sequence Clustering*) algorithm presented in Yang and Gerasoulis (1994).
- **task duplication-based scheduling (TDS)** algorithms, such as presented in Ranaweera and Agrawal (2000)

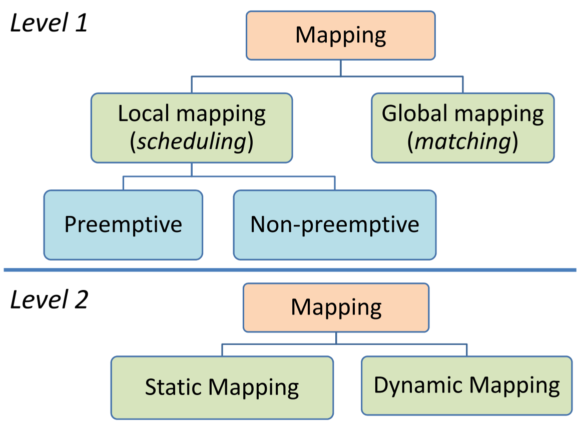

Fig. 1. Classification of mapping levels

The second category, guided random search-based algorithms were explored in several papers. The genetic algorithms are the most used techniques as in Hou et al. (1994) and Lee et al. (1997), indeed they offer a good minimization of the schedule length. However their performance remains unsatisfactory with regard to their run time requirements.

The main observation we made while reviewing the literature is the lack of approaches that yield the exact solution for mapping on heterogeneous architectures. Our work differs from those cited above by focusing especially on mapping tasks onto the heterogeneous CPU/FPGA systems in static and runtime phases, for a nonpreemptive execution of the tasks. In this work, we exploit the heterogeneous CPU/FPGA architecture by using several options for assigning tasks.

3. PROBLEM STATEMENT

We consider the challenge of mapping a simulation project containing several tasks onto a dynamically reconfigurable CPU/FPGA architecture at the initial phase. We denote by C the set of cores and by F the set of FPGAs as illustrated in Figure 2 (left). We denote $\gamma = \mid C \mid$ and $\phi = \mid F \mid$ where $\mid x \mid$ indicates the cardinality of x. In this work C contains four cores $[C1, C2, C3, C4]$ and F contains only one element $[F1]$, hence $\gamma = 4$ and $\phi = 1$.

As described in the example Figure 2 (right), we mapped the project onto an architecture with 4-cores CPU and one FPGA. In this example, tasks of project P1 are sequentially executed respecting the task graph precedence as shown in Figure 2 (right).

A project consists of several tasks with data dependencies among them. Each edge between two tasks i and k represents a precedence constraint, while the edge 'weight' represents the average communication cost between the two tasks. A task without any parent is called an "initial task" and is denoted "ent" (task 1 in the example of Figure 2 (right)), and a task without any child is called a "terminal task" and is denoted "exit" (task 12 in the example of Figure 2 (right)). A task can be executed on a given processor only when all data from its parents become available to that processor.

The following assumptions are made:

- The tasks are numbered topologically, hence all the predecessors of any task have an inferior index to that task.
- Task execution of a given application is non-preemptive.
- A task could have three types of implementation:
    - · Type 1: only a software version on a CPU.
    - · Type 2: only a hardware version on FPGA.
    - · Type 3: both hardware and software versions.

In this section, we first present an exact method to get an optimal mapping of a given project, then we present a heuristic approach in order to improve the speed of performance. For both approaches, we have the following notation in common:

Symbol	Definition
n	number of tasks.
m	number of processors.
C	the set of cores.
F	the set of FPGAs.
$\gamma = \mid C \mid$	the cardinality of C.
$\phi = \mid F \mid$	the cardinality of F.
t_{ij}	the execution time of task i in processor j.
c_{ik}	the average communication cost between task i and task k.

- n and m are, respectively, the number of tasks and the number of processors.

$$m = \gamma + \phi \; where \; \gamma = \mid C \mid \; and \; \phi = \mid F \mid \quad (1)$$

- t_{ij} : the execution time of task i in processor j. Note that j is the index of both the CPU Cores and the FPGA.
- c_{ik} : the average communication cost between i and k

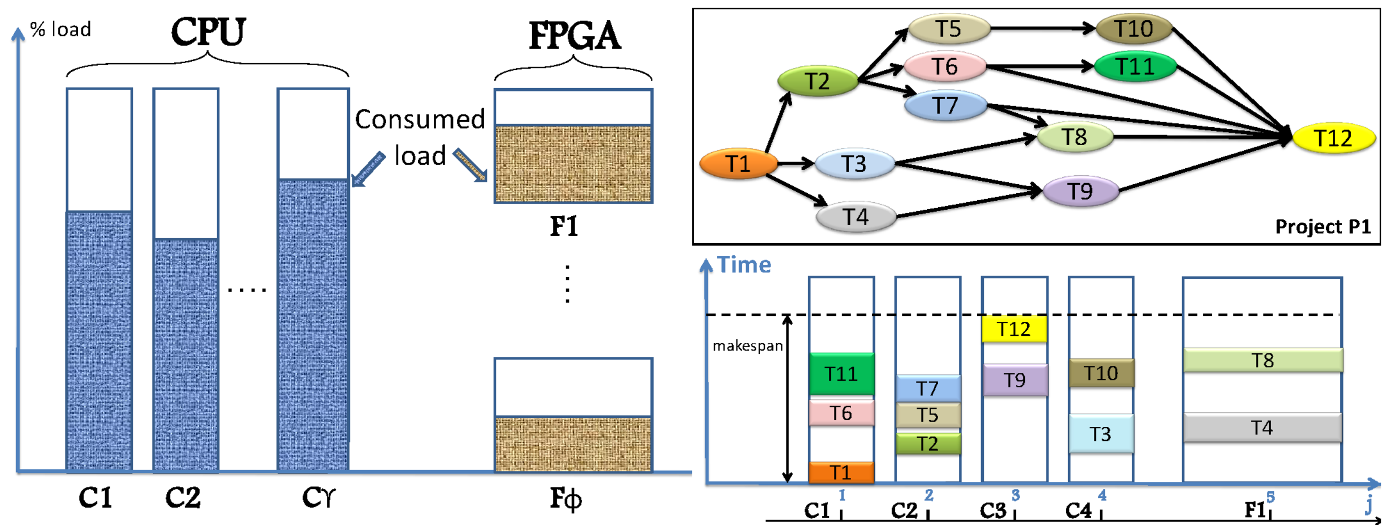

Fig. 2. Occupancy of computing capacity at time t (left). Example of task mapping onto a CPU/FPGA system (right).

4. THE EXACT METHOD

4.1 Notation

Symbol	Definition
ST_i	Start Time of a task i
P_{ki}	binary parameter on communication between two tasks i and k
x_{ij}	binary decision variable to assign a task i to a processor j
a_{ik}	binary variable for non-preemption constraint

To model the exact approach, we introduce the variable ST_i to indicate the Start Time of a task i. The start time of the entry task is $ST_1 = 0$. Note that we will not use a variable to indicate the finish time, since we want to reduce the number of variables and minimize the execution time of the exact mapping algorithm.

The finish time of some task i will be equal to the start time plus the execution time of the task i. $\forall i \in \{1 \cdots n\}$:

$$ST_i + \sum_{j=1}^{m} x_{ij} * t_{ij} \tag{2}$$

In order to specify if there is any communication between two tasks i and k, we define P_{ki} as follows:

$$P_{ki} = \begin{cases} 1, if\ Ti\ and\ Tk\ are\ linked\ by\ precedence \\ \quad and\ Tk\ precedes\ Ti \\ 0, otherwise \end{cases} \tag{3}$$

Note that if $P_{ki} = 1$ we will have $k < i$.

We define the binary variable x_{ij} as the decision variable to assign a task i to a processor j, then:

$$x_{ij} = \begin{cases} 1, if\ task\ i\ is\ allocated\ to\ processor\ j\ in\ C \cup F \\ 0, otherwise \end{cases} \tag{4}$$

4.2 System constraints

- First, each task must be assigned to a processor, where C is the set of CPU Core processors and $|C| = m : \forall i \in \{1 \cdots n\}$

$$\sum_{j=1}^{m} x_{ij} = 1 \tag{5}$$

This equation forces each task to have its $x_{ij} = 1$ for one and only one processor j.

- Secondly, the precedence constraint must be fulfilled: $\forall i \in \{2 \cdots n\}$, $\forall k \in \{1 \cdots i-1\}$

$$ST_i \geq P_{ki} * (c_{ki} + ST_k + \sum_{j=1}^{m} x_{kj} * t_{kj}) \tag{6}$$

This equation forces the start time of a task i "ST_i" to be later than the finish time of all predecessors k of task i "$ST_k + \sum_{j=1}^{m} x_{kj} * t_{kj}$" plus the communication time c_{ki} see Figure 3 . The inequation is taken into account when the task k precedes i, i.e. P_{ki}=1.

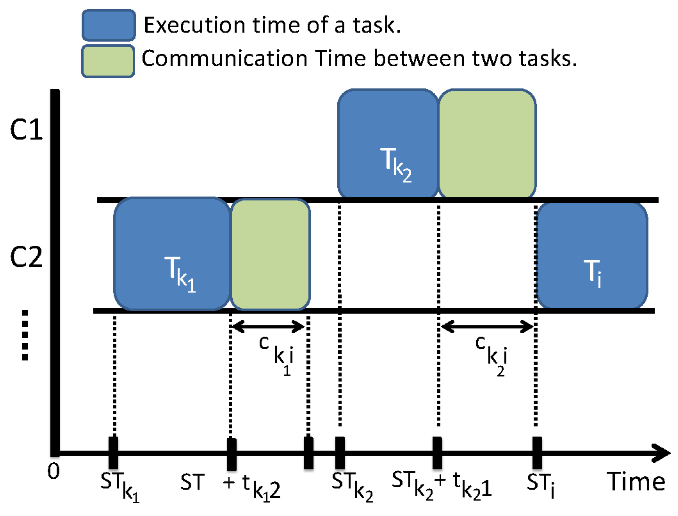

Fig. 3. Example of two tasks k_1 and k_2 linked by precedence to some task i

- Finally, the preemption is not allowed:
  If two tasks, say i and k are assigned to the same processor then they must be separated by the processing time of the earliest one, whether the two tasks are related by precedence or not (In case of precedence the condition is implicitly taken care of in the second constraint). Since we do not know beforehand which task will precede which other task on any processor, let a_{ik} be a binary variable as presented in Figure 4. The no-preemption condition is then provided by the following two inequalities, in which W1 and W2 are two very large numbers.
  $\forall i,k \in \{1 \cdots n\}$, $\forall j \in \{1 \cdots m\}$ and $i < k$

$$\begin{aligned} &ST_i + t_{ij} \leq ST_k + W1 * (2 - x_{ij} - x_{kj}) + W2 * a_{ik} \\ &ST_k + t_{kj} \leq ST_i + W1 * (2 - x_{ij} - x_{kj}) + W2 * (1 - a_{ik}) \end{aligned} \tag{7}$$

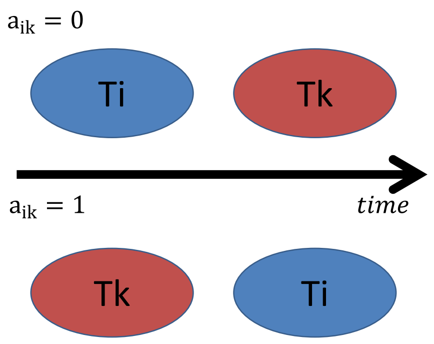

Fig. 4. Definition of the a_{ik} variables

Note that usually the non preemption cosnstraint is formulated as follows:

$\forall i,k \in \{1 \cdots n\}$, $\forall j \in \{1 \cdots m\}$

$$\begin{aligned} &ST_i + t_{ij} \leq ST_k + W1 * (2 - x_{ij} - x_{kj}) + W2 * a_{ik} \\ &a_{ik} = 1 - a_{ki} \end{aligned} \tag{8}$$

With the formulation given in equation 7, we diminished the number of variables and constraints.

4.3 Objective function

We wish to obtain the fastest execution of each project, which amounts to minimize the makespan C_{max}. In our context minimizing the C_{max} is eqquivalent to minimize the finish time of the "terminal task". The objective is realized by adopting the following objective function:

$$min[ST_n + \sum_{j=1}^{m} x_{nj} * t_{nj}] \tag{9}$$

Note that, as explained before equation (2), the expression between brackets in equation (9) is equivalent to the finish time of the terminal task.

5. THE MHEFT HEURISTIC

The proposed algorithm is inspired strongly by the HEFT algorithm (Heterogeneous Earliest Finish Time algorithm). As described in Topcuoglu and Wu (2002), the HEFT is an application scheduling algorithm for a bounded number of heterogeneous processors, which has two main phases:

- Compute the priorities of all tasks;
- Select the tasks according to their priorities, then schedule each task on its "best" processor that minimizes the task finish time.

The HEFT algorithm minimizes the makespan, which is our objective in this study.

In this section, we are given the following information:

Symbol	**Definition**
$pred(i)$	the set of immediate predecessor tasks of task i.
$succ(i)$	the set of immediate successor tasks of task i.
$EST(i,j)$	the Earliest execution Start Time of task i on processor j.
$EFT(i,j)$	the Earliest execution Finish Time of task i on processor j. Note that $EST(ent,j) = 0$, since ent is what we called before, the entry task.
AFT (i)	the Actual Finish Time of the task Ti.
ACC (i)	the Average Computation Cost of a task Ti.

$$EST(i,j) = \max\left(avail[j], \max_{k \in pred(i)} (AFT(k) + c_{ki})\right) \tag{10}$$

Where avail[j] is the earliest time at which processor j is ready for task execution. For example if Tk is the last assigned task on processor j, then avail[j] is the time that processor j completed the execution of the task k and it is ready to execute another task. AFT (k) is the actual finish time of the task k. We deduce that:

$$EFT(i,j) = t_{ij} + EST(i,j) \tag{11}$$

The schedule length of one project of tasks is the AFT of the terminal task of this project. The schedule length which is also called makespan is defined then as follows:

$$makespan = AFT(exit) \tag{12}$$

Given a project with n tasks and m processors, the Average Computation Cost (ACC) of a task (i) is computed by dividing the sum of computation costs of the task on each processor by the number of available processors.

$$ACC(i) = \frac{\left(\sum_{j=1}^{m} t_{ij}\right)}{m} \tag{13}$$

In order to establish the priority of tasks allocation we introduce a new parameter called the upward rank "$rank_u$" , which is defined recursively as follows:

$$rank_u(i) = ACC(i) + \max_{k \in succ(i)} (c_{ik} + rank_u(k)) \tag{14}$$

The rank is computed recursively by traversing the task upward. For the terminal task "exit", the upward rank value is equal to

$$rank_u(exit) = ACC(exit) \tag{15}$$

$rank_u(i)$ is the length of the critical path from the task Ti to the terminal task, including the computation cost of task Ti.

The two steps of the *Modified* HEFT algorithm are then, as follows:

- Compute the priorities of all tasks:
  The priority is established using the upward rank value $rank_u$. The list of priorities is generated by sorting the tasks by decreasing order of $rank_u$.
- Best processor selection phase:
  The HEFT algorithm has an insertion-based policy which considers the possible insertion of a task in an earliest idle time slot between already-scheduled tasks on a processor. The length of an idle time-slot, i.e, the difference between execution start time and finish time of two tasks that were consecutively scheduled on the same processor, should be at least capable of computation cost of the task to be scheduled. Scheduling on this idle time slot should preserve precedence constraints.

Algorithm 1 Algorithm MHEFT

```
Set the computation costs of tasks and communication costs between tasks.
Compute rank_u for all tasks by traversing graph upward, starting from the terminal task.
Sort the tasks in a scheduling list by nonincreasing order of values.
while there are unscheduled tasks in the list do
  Select the first task Ti, from the list for scheduling.
  if Ti ∈ Type 1 then
    for each core j in the core-set (j ∈ C) do
      Compute EFT (i, j) value using insertion-based scheduling policy.
      Assign task Ti to the core j that minimizes EFT of task i.
    end for
  else if Ti ∈ Type 2 then
    for each FPGA j in the FPGA-set (j ∈ F) do
      Compute EFT (i, j) value using insertion-based scheduling policy.
      Assign task Ti to the FPGA j that minimizes EFT of task Ti.
    end for
  else if Ti ∈ Type 3 then
    for each processor j in the processor-set (j ∈ C U F) do
      Compute EFT (i, j) value using insertion-based scheduling policy.
      Assign task Ti to the processor j that minimizes EFT of task Ti.
    end for
  end if
end while
```

For recall definition of Type 1, 2 and 3, see problem statement in Section 3.

6. THE MHEFT HEURISTIC VS THE EXACT METHOD

In this section we present a comparison between the MHEFT Heuristic and the Exact method. In order to compare the performance of each method we generate several graphs containing from 2 to 40 tasks. The comparison metrics are the makespan, the relative gap of the makespan and the mapping execution time. The relative gap of the makespan is calculated by the following ratio:

$$\frac{makespan_{MHEFT} - makespan_{Exactmethod}}{makespan_{Exactmethod}} \cdot 100 \qquad (16)$$

While analysing the makespan obtained by both MHEFT Heuristic and Exact method, we observe that the MHEFT Heuristic gives a good performance, indeed, the makespan is close or equal to the makespan obtained with the Exact method. However, there are some problematic cases where the relative gap is greater than 10%, a preliminary analysis shows that it is due to several parameters such as the ratio = number of precedence links / number of tasks. We want to improve in future works the MHEFT Heuristic in order to stabilize the relative gap at a value less than or equal to 10%. For the mapping execution time we obtained a result less than 0.5 ms for all the instances with the MHEFT Heuristic. Table 1 gives the execution times obtained with the exact method. As expected the MHEFT Heuristic is much faster than the exact method.

Instance size	Execution time
[2...10]	≤ 0.04s
[11...20]	≤ 0.35s
[21...30]	≤ 9.25s
[31...40]	≤ 35.71s

Table 1: Execution times obtained with the exact method

7. CONCLUSION AND PERSPECTIVES

In this paper we studied the problem of mapping a project of several tasks into the heterogeneous architecture CPU/FPGA. This problem is proven to be NP complete, see Fernandez-Baca (1989) and Garey and Johnson (1990). In spite of this complexity, a mathematical model solved by an exact method has been proposed and evaluated. In practice, the experience shows that the use of an exact method remains very important in the context of real time constraints. Indeed, if we assume a simulation test which is executed on several hours, even if the execution of the exact method may take several minutes, this, is negligible compared to an overload which may cause the loss of the entire test simulation. A good initial mapping will permit to avoid such overload risks.

In this study, average communications costs were considered between each pair of tasks linked by precedence, and the execution, into processors was considered as non preemptive. We first presented an exact method which we will use for the static initial mapping, furthermore, it serves as a reference, for evaluating the optimality of heuristics approaches. The second approach proposed in this work is a modified version of the heuristic “HEFT”. The changes concern the limitation of options choice. Since in this part, some tasks are exclusively executable on some kind of processors. The MHEFT gives a good optimality and will serve to assign new arrivals tasks at runtime phase. Also the MHEFT results represent an upper bound for the exact method, indeed, this information may permit to speed up the execution time of the exact method.

In order to enrich this work, in future research, we will compare the performance of our approaches with the achievement of several experiments. Also we look to consider the exact communication costs instead of the average in order to improve the optimality. Another possible extension of the settings studied in this paper is to consider that the execution time of tasks may increase during the execution of a project of tasks so as to be more realistic. In order to speed up the execution time of the exact method, additional weights for the selection of the processors can be an efficient solution to explore, indeed, the mathematical model presents an important number of symmetries, since we consider several identical cores. We plan to extend our algorithms to deal with the multi-objective "makespan minimization" and "load balancing". Finally, we intend to investigate the dynamic mapping in order to deal with re-allocation of tasks so as to anticipate the onset of overloads.

REFERENCES

Alaoui, S.M., Frieder, O., and El-Ghazawi, T. (1999). A Parallel Genetic Algorithm for task mapping on parallel machines. In *Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, 201–209.

Asano, S., Maruyama, T., and Yamaguchi, Y. (2009). Performance Comparison of FPGA, GPU AND CPU in Image Processing. In *19th IEEE International Conference on Field Programmable Logic and Applications, FPL*, 126–131.

Bck, T. (ed.) (1997). *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, USA, East Lansing, Michigan, USA. ISBN: 1-558-60487-1.

Braun, T.D., Siegel, H.J., and Beck, N. (2001). A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. 61, 810–837.

Casavant, T.L. and Kuhl, J.G. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. 14, 141–154.

Chen, W.H. and Lin, C.S. (1998). A hybrid heuristic to solve a task allocation problem. In *Computers and Operations Research*.

Fazlali, M., Sabeghi, M., Zakerolhosseini, A., and Bertels, K. (2010). Efcient task scheduling for runtime recongurable systems. 56, 623–632.

Fernandez-Baca, D. (1989). Allocating modules to processors in a distributed system. *IEEE Transactions on Software Engineering*, 15, 1427–1436.

Garey, M.R. and Johnson, D.S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

Hou, E.S.H., Ansari, N., and Ren, H. (1994). A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5, 113–120.

Ibarra, O.H. and Kim, C.E. (1977). Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the Association for Computing Machinery*, 24, 280–289.

Jong-Kook, K., Shivle, S., Siegel, H.J., Maciejewski, A.A., Braun, T.D., Schneider, M., Tideman, S., Chitta, R., Dilmaghani, R.B., Rohit, J., Aditya, K., Ashish, S., Siddhartha, S., Vangari, P., and Yellampalli, S.S. (2007). Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *Journal of Parallel and Distributed Computing*, 67, 154–169.

Koulamas, C. and Kyparisis, G.J. (2007). An improved delayed-start LPT algorithm for a partition problem on two identical parallel machines . In *European Journal of Operational Research*, volume 187, 660–666.

Lee, W., Jay, S.H., Roychowdhury, V.R., and Maciejewski, A.A. (1997). Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 47, 8–22.

Lin, S.C., Goodman, E.D., and Punch, W.F. (1997). A genetic approach to dynamic job shop scheduling problems. In *Bck (1997)*, 481–488.

Luo, P., Lu, K., and Shi, Z. (2007). A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. 67, 695–714.

Ranaweera, S. and Agrawal, D.P. (2000). A task duplication based scheduling algorithm for heterogeneous systems. *International Parallel and Distributed Processing Symposium (IPDPS'00)*, 14, 445–450.

Shoukat, A., Jong-Kook, K., Siegel, H.J., Maciejewski, A.A., Yu, Y., Gundala, S.B., Gertphol, S., and Prasanna, V.K. (2002). Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 2, 519–530.

Topcuoglu, H. and Wu, M.Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13, 260–274.

ulrich Heiss, H. (1992). Mapping tasks to processors at runtime. In *International Symposium on Computer and Information Sciences*, 515–518.

Yang, T. and Gerasoulis, A. (1994). Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5, 951–967.