Task Placement Heuristic Based on 3D-Adjacency and Look-Ahead in Reconfigurable Systems

Jesús Tabero

Dept. Programas Espaciales
Instituto Nacional de Técnica Aeroespacial
Madrid 28850, Spain
Tel : +34-91-520-1693
Fax : +34-91-520-1492
taberogj@inta.es

Julio Septién, Hortensia Mecha, Daniel Mozos

Dept. Arquitectura de Computadores y Automatica
Universidad Complutense de Madrid
Madrid 28040, Spain
Tel : +34-91-394-7617/18/19
Fax : +34-91-394-7527
{jseptien,horten,mozos}@dacya.ucm.es

Abstract- To get efficient HW management in 2D Reconfigurable Systems, heuristics are needed to select the best place to locate each arriving task. We propose a technique that locates the task next to the borders of the free area for as many cycles as possible, trying to minimize the area fragmentation. Moreover, we combine it with a look-ahead heuristic that allows delaying the scheduling of a task to the next event, increasing the solution search space.

I. Introduction

Current multimedia systems have a very dynamic behaviour. Moreover, sometimes the user wants to upgrade their functionality by loading new applications. Traditional platforms based on processors are not able to provide the needed performance, whereas ASICs lack flexibility to deal with such a changing environment.

A technology that lies between processors and ASICs is reconfigurable hardware. It participates of the flexibility of processors and, at the same time, it is able to offer very good results in terms of performance. In recent years this flexibility has increased with the possibility of partially reconfiguring the hardware at run time [1]. This allows changing the functionality of a digital system under user demand, as multimedia applications require. It also allows true hardware multitasking through space multiplexing.

In order to use all these capabilities, the functionality of the operating system must be increased to manage this kind of resources. In [2] some of the main problems of designing such an operating system are outlined.

One of the most interesting problems is to decide where to locate the bitmap of a new task in the FPGA when it must be run. A data structure is needed that maintains information about the available free area, and the algorithm must choose the best place to locate the arriving task, trying to use the reconfigurable area as efficiently as possible.

Our algorithm chooses the best location for an arriving task trying to minimize the fragmentation that this mapping will produce. The fragmentation of the free area is an indirect measure of the probability of finding a suitable location for a new task in the FPGA in the near future.

The rest of the paper is organized as follows. Section II reviews related work in this field. Section III presents the algorithm for area management. Section IV describes the heuristics used to select a location for an arriving task. And finally, in sections V and VI some experimental results and conclusions are presented.

II. Related work

Task allocation, free space management and area fragmentation are fundamental problems of managing 2D reconfigurable resources. They have been dealt with recently by several research teams.

Diessel et al. [3] have developed a quad-tree structure to store the information of the available FPGA area. Such structure can be travelled and updated quite fast, but it does not guarantee that an adequate place is found, even if there is enough area to store the task, but split among different branches of the tree. This solution does not take into account the resultant free area fragmentation to select the position where the task is mapped to. On the contrary, it deals with fragmentation by proposing several high-cost defragmentation processes.

Bazargan et al. [4] deal with the area allocation problem by using a bin-packing approach and applying some of the classical algorithms for such theoretical problem. They propose several strategies for on-line 2D bin-packing of the arriving rectangular tasks. These strategies differ mainly in the way the free area is managed. One of them keeps track of all the maximum empty rectangles (MER) where an arriving task could be placed. Such approach guarantees that, if an adequate place exists, it can be found, but at the cost of a very high complexity. A second approach tries to use heuristics in order to reduce the number of rectangles considered when updating the rectangle list. When a free rectangle is selected to store the arriving task, the excess area is divided in only two, non-overlapping, new rectangles. Bazargan offers several criteria to do this splitting, but does not decide clearly for one of them. Anyway, by selecting some of the possible rectangles, situations can arise where existing room cannot be used to store a task, because it is split among several rectangles.

Walder et al. propose in [5] an enhanced version of Bazagan's partitioner with the same efficiency but improved placement quality. This enhanced method delays the basic vertical/horizontal split decision and manages overlapping rectangles in a restricted form. They also present a hash matrix approach to find a placement in constant time, but the updating of this structure is very time consuming.

Ahmadinia et al. [6] present another version of Bazagan´s partitioner, managing the occupied area instead of the free area, so in most cases the number of rectangles is much lower, though the complexity order is the same. The heuristic used to allocate a task tries to minimize the distance to the tasks it communicates with. But it does not take into account the area fragmentation, nor time or data constraints during scheduling.

Handa et al. [7] present a fast algorithm for finding empty space in a FPGA which uses a staircase data structure to report the empty area in the form of a list of MER. The search of such structure has a complexity order of O(m*n) where m and n are the number of columns and rows respectively, but there is no details of the MER selection criterion, and the complexity of the task placement algorithm is not considered.

III. Vertex-List Based Area Management

Our approach to reconfigurable HW management keeps track of the available FPGA free area with a vertex-list structure that has been described in detail in [8]. Such structure can be travelled with different heuristics in order to choose the vertex where the arriving tasks will be placed.

As fig.1 shows, we use a 2D FPGA model, an homogeneous two dimensional grid formed by W*H basic reconfigurable blocks, that we will use as "area units" all along. We suppose that each basic block includes processing elements as well as I/O resources. The FPGA has also some dedicated resources to manage the task I/O. A task can be made of an arbitrary number of such basic blocks, but always with a rectangular shape. The tasks are relocatable and can be inserted at arbitrary row and column offsets. The tasks are independent, with no data constrains between them, but there can be real-time constrains that must be satisfied.

Each task is defined by the following parameters: $T_i = \{ w_i, h_i, t\_ex_i, t\_arr_i, t\_max_i \}$, where w_i and h_i indicate the task size, t_ex_i is the task execution time, t_arr_i the task arrival time and t_max_i the maximum time allowed for the task to finish execution. Therefore, the task T_i can't be scheduled later than $t\_max_i - t\_ex_i$.

Fig. 1 summarizes the operation of our HW manager, that consists of three main modules, the Task Scheduler, the Vertex Selector and the Vertex List Updater, and uses three

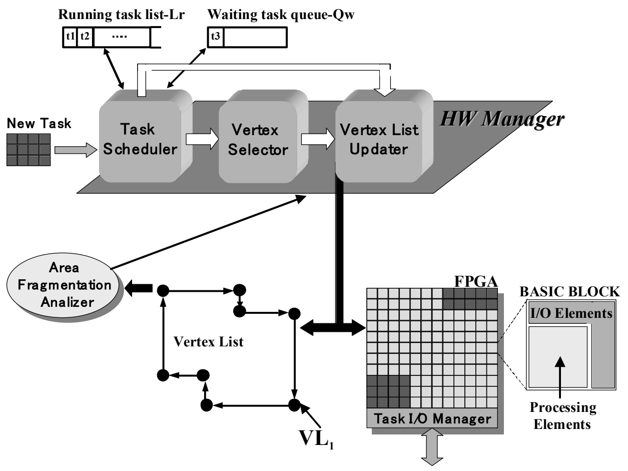

Fig. 1. HW manager and vertex-list structure

important data structures, the Running Task List, L_r, the Waiting Task Queue, Q_w, and the Vertex List Set, VLS, that describes all the available FPGA free space, with a different Vertex-List VL_i for each FPGA free hole.

The Free Area Fragmentation Analizer is a module that computes the fragmentation of the FPGA free area for a given FPGA status, by using a new fragmentation metrics described in detail in section IV. This value can be used either by the HW Manager to perform a defragmentation process, or by the Vertex Selector Module if a fragmentation-based heuristic is used.

When a new task (an arriving or waiting task) is considered, the Task Scheduler calls the Vertex Selector to check whether a feasible position exists where the task could be mapped to. The vertex is chosen among the candidates according to the selected heuristic. The candidates can be of several types: bottom-left (BL), bottom-right (BR), top-left (TL) or top-right (TR). Then the task is inserted and the VLS is updated by the Vertex List Updater accordingly. When a task ends, the VLS is also updated, and different special situations such as hole merging or island managing, that are described in detail in [8], must be dealt with. If there is no place for an arriving task, the task is temporally stored at the queue Qw, and if timeout happens it is discarded.

IV. Heuristics for Location Selection

An heuristic is used to choose a given vertex among all the feasible candidates to locate the task. A simple approach based on a First-Fit criterion was proposed in [8]. A more efficient alternative was presented in [9], with a Best-Fit approach based on a cost function computed with a task-adjacency criteria. We also presented a fragmentation metric that was used to evaluate the quality of the FPGA status at any given time. A 2D-Adjacency example is shown in Fig. 2a.

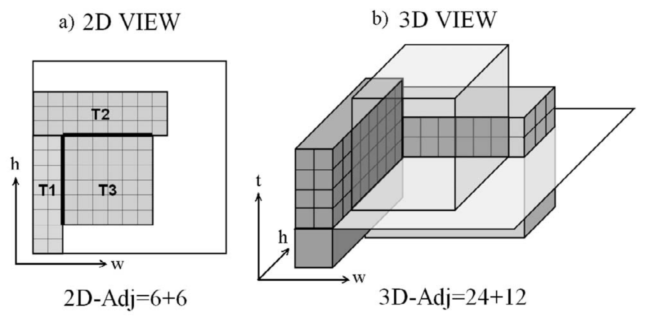

Fig. 2. 2D-Adjacency (a) and 3D-Adjacency (b)

What we are proposing now is a new alternative, where the cost function considers not only 2D-spatial adjacency, but also temporal adjacency. Then we will develop a look-ahead heuristic based on this 3D-adjacency cost function that will give even better experimental results.

A. 2D-Adjacency Heuristic

This heuristic inserts the task at the vertex position where the arriving task achieves the higher contact level between the new task borders and the envelope defined by the VL_i. The adjacency is computed in terms of block-length units. This heuristic was presented in [9], and an example is shown in Fig. 2a, where the adjacency value obtained when T3 is placed next to T1 and T2 is of 12 units.

B. 3D-Adjacency Heuristic

This heuristic is an evolution of the 2D-Adjacency heuristic described above, but extended to the time axis. Considering the FPGA as a bin and the tasks as smaller boxes, with time as vertical axis, the algorithm tries to pile up the new box as close as possible to the rest of boxes already in the FPGA or to the FPGA borders. This heuristic is similar to the one we would use in the real world to place the boxes inside the bin. This approach is shown in Fig. 2b, where the 3D-adjacency value obtained when T3 is placed next to T1 and T2 is now of 36 units (6x4+6x2).

To take this into account, it is necessary to store a new value for each edge of the vertex list: the remaining execution time of the task the edge belongs to, a kind of "edge lifetime". In order to simplify, if an edge belongs to several adjacent tasks, the one with the shortest remaining time is chosen as lifetime for the whole edge.

The 3D adjacency is then computed as the sum of the length of the new task in contact with each adjacent edge, multiplied by the temporal adjacency between the new task and the edge. This temporal adjacency is the minimum between the edge's lifetime and the t_ex of the arriving task. When the arriving task is in touch with any edge of the FPGA perimeter, then t_ex of the arriving task is considered. Thus, when it is possible at the current time, the task will be preferably placed next to FPGA borders, or to other long-life tasks.

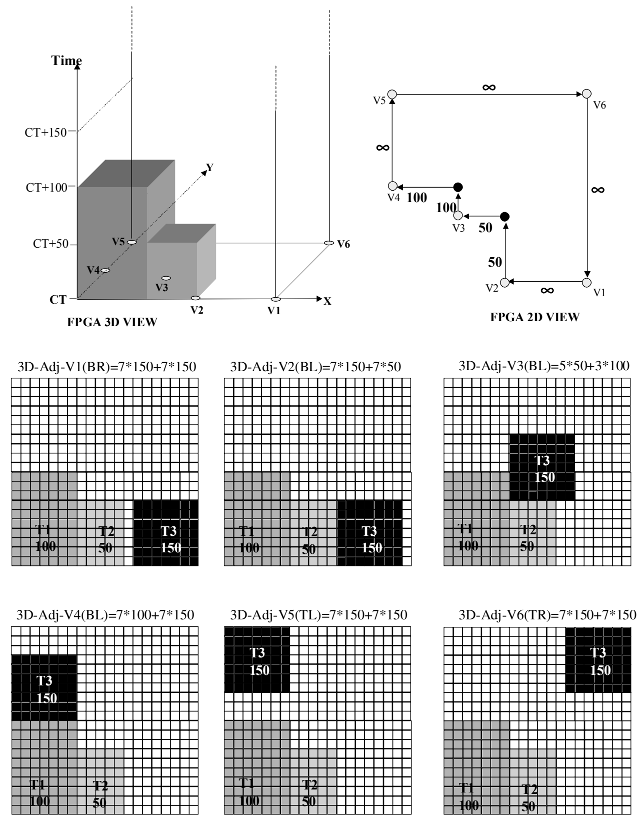

Fig. 3. 3D-Adjacency Heuristic computation

Fig. 3 shows this concept using an FPGA with two running tasks T1 and T2. When T3 arrives, with a size of 7*7 basic cells and a t_ex value of 150 time units, the HW Manager tries to schedule it at CT (Current Time). The 3D-adjacency value for all the candidates vertex is calculated as it is shown in the figure. Therefore the new task T3 will be inserted on the candidate vertex V1, the first one where the 3D-adjacency value is maximum.

Fig. 4 shows a simple example to illustrate the behavior of the heuristic. The FPGA initial status is shown on top. There are four currently running tasks, with their remaining execution time shown inside, and the corresponding VL. When a new task, of 3*4 basic cells and a t_ex value of 150 time units arrives, this approach computes the 3D-adjacency value for each feasible candidate position. As it can be seen, this heuristic would place the task at candidate A (B is equivalent).

We can analyse the quality of the decisions taken by the 3D-Adjacency heuristic by using the fragmentation metric proposed in [8], where the fragmentation level of the free area for a given FPGA status was estimated as follows:

$$F = 1 - \Pi_i \ [(4/V_i) * (A_i/A_F)]$$

where the term between brackets represents a kind of suitability for each free hole i, with area A_i and V_i vertices, to accommodate future tasks, while A_F stands for the whole free area in the FPGA.

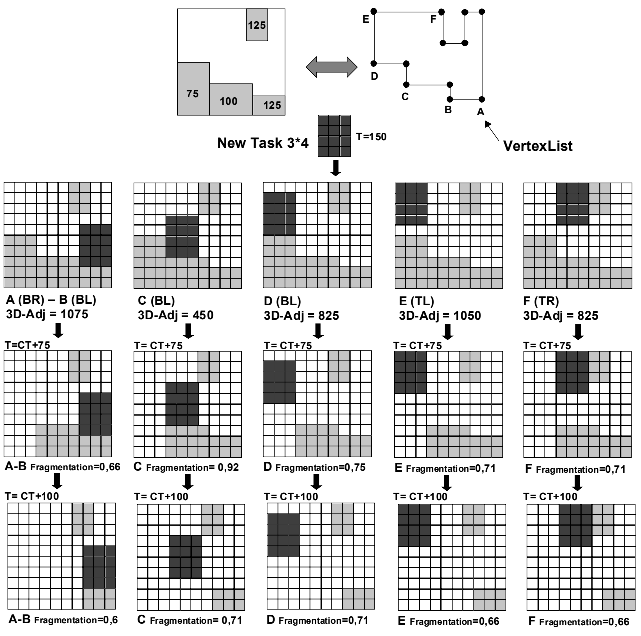

Fig. 4. 3D Adyacency heuristic example

The task insertion at candidate A, the one selected by the 3D-Adjacency heuristic, leads to future FPGA states with lowest fragmentation values, according to our metric, at different simulation steps. When currently running tasks finish execution at T=CT+75 and at T=CT+100, the fragmentation obtained for candidate A is the lowest, as it can be seen in fig. 4 where this fragmentation value is labelled at the bottom of each feasible candidate position.

C. Look-ahead 3D Heuristic

This heuristic uses the 3D-adjacency value, computed as it was described earlier, but in this approach this value is calculated for all feasible candidates at both the current simulation time and at the next event time (when the next task-end happens), performing thus a one-level look-ahead scheduling. The idea is that we could get a better 3D-adjacency value for a location only available in the near future.

However, sometimes it is not possible to perform this look-ahead calculation, if the new task must be scheduled before any running task can exit the FPGA, due to its timeout t_max$_i$.

Fig.5 shows a comparative between 3D-Adjacency (3D-Ad) and Look-Ahead-3D (LA-3D) heuristics behavior, when two new tasks, T3 and T4, arrive simultaneously.

We have considered a 20*20 FPGA, with two currently running tasks defined by the tuple of parameters (described in section III): T1={12,12,6,98,110} and T2={2,13,15,98,118}, placed at bottom-left and top-right FPGA corners respectively.

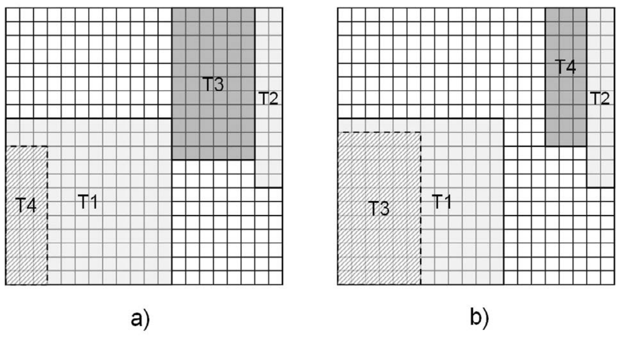

Fig. 5. a) 3D-Ad Heuristic b) LA-3D Heuristic

Let us suppose T3={6,11,8,100,120} is processed then. The 3D-Ad heuristic would give at current simulation time, CT=100, a maximum value for TR candidate vertex at coordinates X=18, Y=20. In the other hand, the LA-3D heuristic would also compute the 3D-adjacency value at the next event time, and would place T3 at the BL candidate at coordinates X=0, Y=0, at Time=104 when T1 finishes execution (fig. 5b).

When T4={3,10,21,100,123} is processed, at CT=100 also, the 3D-Ad heuristic can not find a feasible candidate and T4 would be rejected at Time =103 because of its t_max value. On the contrary, LA-3D heuristic would find a feasible candidate for T4 at X=18, Y=20 which satisfies its real-time constrains.

V. Experimental Results

To evaluate the quality of our new approaches, we have made experiments using four different algorithms for area management:

a. *Classical FF with BL heuristic(FF_BL).* When a new task arrives to the FPGA, it performs an exhaustive search through the block matrix, from left to right and from bottom to top, in order to find a feasible location for the arriving task.
b. *Vertex List with BF_2D-Adjacency heuristic (2D-Ad).* Presented in [9].
c. *Vertex List with BF-3D-Adjacency heuristic (3D-Ad).*
d. *Vertex List with BF_LA-3D-Adjacency heuristic (LA-3D).*

These four algorithms have been tested with a simulated FPGA of 100*100 basic blocks, and different data sets of 100 tasks each. They have been randomly generated with a task size range and ratio, similar to others found in many multitasking environments. Table 1 shows these data sets which have been classified in three classes, considered as representative scenarios, depending on the task size ranges and their temporal features.

TABLE I
Data set classes

Data Sets	Min. task area	Max. task area	Data Set features
D1, D2	5*5	40*40	Low arrival frequency Low execution time Low task size
D3, D4	10*10	50*50	Medium arrival frequency Medium execution time Medium task size
D5, D6	10*10	60*60	High arrival frequency High execution time High task size

We have used two different parameters to evaluate the results obtained. Table 2 includes for each data set, the percentage of the computing volume rejected for each algorithm. This volume represents all the tasks that were rejected because the manager was not able to find a proper location in time to meet the task's time constraint. For each task, the volume is the product of the task area multiplied by its execution time. The other parameter shown in table 2 is the average FPGA occupation maintained by each algorithm when processing the different data sets.

Figure 6 shows the detailed and the average computing volume rejected by the four algorithms, respectively. 3D-Adjacency and LA-3D-Adjacency show better performance than the others. Moreover, LA-3D is able to execute all the tasks, giving a zero value for these parameters in all data sets. Also LA-3D produces the best average occupation level as it is shown in fig. 7.

TABLE II
Experimental results

Data Set	% Computing Volume Rejected				% Average Occupation Level			
	FF-BL	2D-Adj	3D-Adj	LA-3D-Adj	FF-BL	2D-Adj	3D-Adj	LA-3D-Adj
D1	8,4	1,5	0	0	22,6	24,5	24,9	24,9
D2	8,5	5,8	2,4	0	28,8	29,6	30,7	31,4
D3	4,5	7,8	4,3	0	35,2	32,9	35,2	36,6
D4	9,5	2,2	5,6	0	32,9	35,05	34,2	36,1
D5	3,32	11,5	0	0	38,7	34,2	40	40
D6	14,9	14,5	1,9	0	41,4	42,1	48	48,7

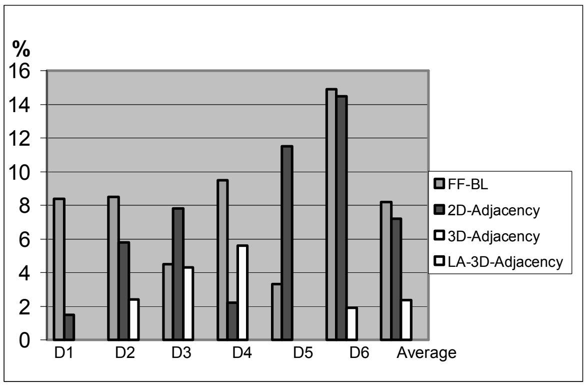

Fig. 6. Detailed and Average Computing Volume Rejected (Percentage)

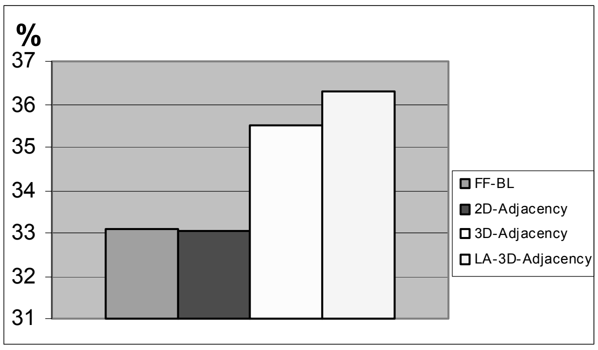

Fig. 7. Average FPGA Occupation Level (Percentage)

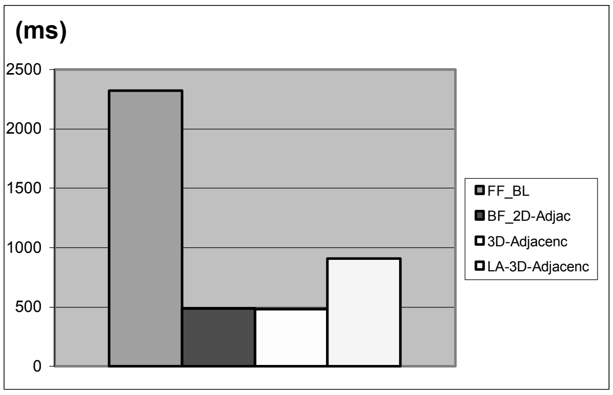

Fig. 8. Relative Average Execution Times

Finally, fig. 8 shows a comparative of the average execution time of the algorithms for the different heuristics. Logically 2D and 3D-Adjacency present the same values because they have the same complexity, and LA-3D takes almost twice this time, because it has to perform the same computation at current time and at the next event time. Finally FF_BL, being the algorithm with a higher complexity order, spends much more time than the others.

VI. Conclusions

We have presented a new approach to HW multitasking allocation, with an area manager that maintains the free area with a vertex-list structure. As an adequate management of the fragmentation problem has revealed itself crucial, our heuristics tries to keep it as low as possible. This is done by allocating the tasks next to the borders of the free area for as many cycles as possible. This heuristic has produced very good results in terms of FPGA area occupation and number of task executed. Finally, we have proposed a look-ahead heuristic that allows delaying the allocation of a task to the next event in order to increase the solution search space. It has clearly shown the best behavior.

Acknowledgements

This work has been supported by Spanish Government research grant TIC2002-00160.

References

[1] K. Compton, S. Hauck, "Reconfigurable computing: a survey of systems and software", ACM Computing Surveys, Vol. 34, No. 2, pp 171-210. June 2002.

[2] O. F. Diessel, G. Wigley, "Opportunities for Operating Systems Research in Reconfigurable Computing", Technical Report ACRC-99-018. Advanced Computing Research Centre, School of Computer and Information Science, University of South Australia, 1999.

[3] O. F. Diessel, H. Elgindy, "On dynamic task scheduling for FPGA-based systems", International Journal of Foundations of Computer Science, IJFCS'01, Vol. 12, No. 5, 2001.

[4] K. Bazargan, R. Kastner, M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems", IEEE Design and Test of Computers, Vol. 17, pp 68–83, 2000.

[5] H. Walder, C. Steiger, M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-Hashing", IPDPS-2003 (RAW'03), vol. 00, p. 178b, Munich, Germany, April 2003.

[6] A. Ahmadinia, C. Bobda, M. Bednara, J. Teich, "A new approach for on-line placement on reconfigurable devices", IPDPS-2004 (RAW'04), vol. 04, nº4, p. 134a, 2004.

[7] Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement," Design Automation Conference, p. 960-965, San Diego, CA, June 2004.

[8] J. Tabero J. Septién, H. Mecha, D. Mozos, "A vertex-list approach to 2D HW multitasking management in RTR FPGAs", DCIS 2003, Ciudad Real, Spain, pp. 545-550, Nov 2003.

[9]. J. Tabero J. Septién, H. Mecha, D. Mozos, "A low fragmentation heuristic for task placement in 2D RTR HW management", FPL 2004, Antwerp, Belgium, pp. 241-250, Sep 2004