Intelligent Merging Online Task Placement Algorithm for Partial Reconfigurable Systems

Thomas Marconi, Yi Lu, Koen Bertels, Georgi Gaydadjiev
Computer Engineering Laboratory, EEMCS
TU Delft, The Netherlands
http://ce.et.tudelft.nl
Email:{thomas,yilu}@ce.et.tudelft.nl, k.l.m.bertels@tudelft.nl, g.n.gaydadjiev@ewi.tudelft.nl

Abstract

Speed and placement quality are two very important attributes of a good online placement algorithm, because the time taken by the algorithm is considered as an overhead to the application overall execution time. To solve this problem, we propose three techniques: Merging Only if Needed (MON), Partial Merging (PM), and Direct Combine (DC). Our IM (intelligent merging) algorithm uses dynamically these three techniques to exploit their specific advantages. IM outperforms Bazargan's algorithm as it has placement quality within 0.89% but is 1.72 times faster.

1. Introduction

One essential problem in partially reconfigurable computing is to search the best way to place tasks on the right position and at the right time on an FPGA in the shortest possible time. The algorithms usually trade off between placement quality and execution speed. Algorithm execution time is, for instance, very important in multithreaded applications in which the flow of code cannot be determined beforehand. Speed and placement quality are therefore two very important attributes of a good online placement algorithm.

One of the dominant approaches is described in [2] where Bazargan et al. presented the Keeping Non-overlapping Empty Rectangles algorithm for online placement. The main disadvantage is that the number of empty rectangles produced by Bazargan's algorithm quickly increase with more task insertions. The algorithm execution time depends on the total number tasks placed on the FPGA and it is dominated by the time for merging and splitting these empty rectangles. In return, Bazargan's algorithm has a high algorithm execution time. The goal of our algorithm is to reduce the execution time of this algorithm while preserving its placement quality.

In this paper, we will compare the algorithm performance based on the following criteria. We define the algorithm execution time as the time needed by the algorithm for a single task placement. Percentage of accepted tasks is the ratio between the total number of accepted tasks and the total number of tasks. Good placement quality algorithms have higher percentage of accepted tasks in general.

This paper presents one solution for prohibitively long algorithm execution times during online placement. The main contributions of this paper are:

- novel three techniques to speedup online placement algorithms;
- a novel Intelligent Merging(IM) online placement algorithm to speedup a good quality well known algorithm.

The remainder of this paper is organized as follows. In Section 2, we discuss related work in online placement algorithms. Detail of our proposed techniques and IM algorithm for online placement are presented in Section 3. In Section 4, we present the evaluation of the proposed techniques and algorithm. Finally, in Section 5, we summarize the paper.

2. Related work

Many authors have already proposed algorithms with shorter execution times compared to Bazargan's [2] proposal. Instead of partitioning the free area, the Vertex-list [8, 9] algorithm uses only the free area perimeter while placing a new task. However this algorithm spends significant time on computing the contact or fragmentation level for placing tasks on one of the corners of this free area perimeter.

Steiger et al. [7] and Walder et al. [12] proposed the On The-Fly (OTF) algorithm by delaying the split decision of

978-3-9810801-3-1/DATE08 © 2008 EDAA

Bazargan's algorithm until the next task is placed on the FPGA to avoid wrong split decision. However they both need to resize several rectangles when a new task is inserted. This process ultimately results in additional execution time.

In [1], Ahmadinia et al. proposed the Horizontal Line (HL) algorithm. Instead of managing a list of empty rectangles, HL uses exactly two horizontal lines for placing tasks; one above (top HL) and one under (bottom HL) the already placed tasks.

In [5] and [6], Handa et al. proposed the Staircase algorithm for finding Maximal Empty Rectangles (MERs). They used a 2D array (referred as area matrix) for modeling the FPGA surface. The area matrix is used for constructing staircases and finally these staircases are used for finding MERs. Hence the bottleneck is the time for constructing staircases and finding MERs.

In [3], Jin Cui et al. proposed the Scan Line Algorithm (SLA). They use the same area matrix as Staircase algorithm in different encoding of the reconfigurable area. In SLA, the area matrix is used for finding Maximum Key Elements (MKEs) and finally these MKEs will be used for finding MERs. Consequently the bottleneck is the time for finding MKEs and MERs.

In [4], Jin Cui et al. proposed the Cell Fragmentation (CF) algorithm. CF uses SLA to find MERs and Fragmentation Matrix (FM). To place a new task on the FPGA, CF needs to compute the Time-Averaged Area Fragmentation (TAAF) for all MERs that is computational intensive.

In [10] and [11], Tomono et al. proposed an algorithm that uses the same area matrix of Staircase algorithm with additional I/O communication constraints. Because of this the status of each communication channel is monitored during the process of creating staircases. Since this algorithm optimizes for short communication distances among placed tasks, additional execution time is required.

3. Our techniques and algorithm

3.1. MON technique

Merging only if needed (MON) is a technique where Non-overlapping Empty Rectangles (NERs) are merged only if there is no available NER for placing the new arrived task. By doing so we can save algorithm execution time (the original algorithm merges NERs always). Figure 1 shows how our MON technique works.

The top left corner of Figure 1 depicts the empty FPGA model (the beginning status) that consists of a single NER (NER A). If there is a new task (T1), the task is placed on the NER A. This process produces two new NERs (B and C) as shown on top right of the same figure. The bottom left of Figure 1 shows the FPGA area when the task T1 is removed from the FPGA after completion, leaving one new NER (NER D). In this situation, Bazargan's algorithm works differently as it would merge the NERs (NERs B, C, and D) into one single bigger NER (NER A in our example). Hence Bazargan's algorithm spends computational time on (unnecessary) merging every time a task completes. In case of MON when a new task (T2) arrives, it is placed on one of the available NERs (in our example NER C) that has enough size to accommodate it. Reducing the unnecessary merging is the key factor in our MON technique for improving the Bazargan's algorithm execution time.

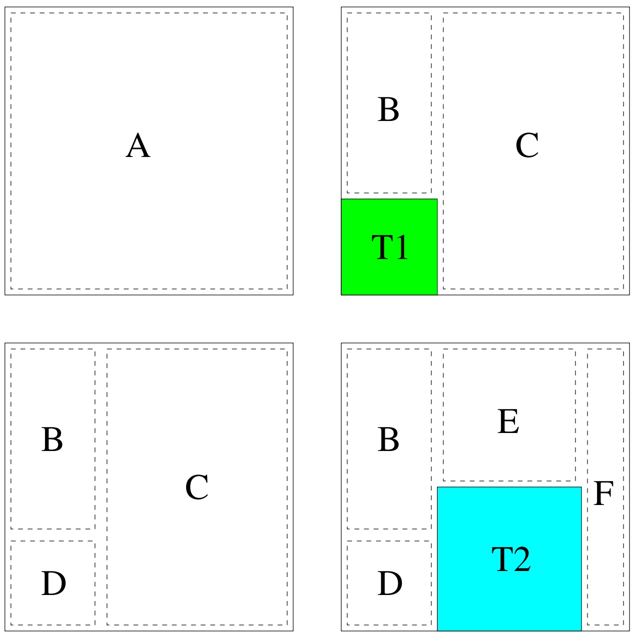

Figure 1. MON technique

3.2. PM technique

Partial Merging (PM) technique allows our Intelligent Merging mechanism to merge only a subset of the available NERs until there is enough free space for the new task. We thus again save algorithm execution time by terminating the merging process earlier. In Bazargan's algorithm as mentioned earlier all available NERs will be merged. Figure 2 shows how this PM technique works.

Top left of Figure 2 shows how three tasks (tasks T1, T2, and T3) have been placed on the FPGA. Task T2 produces two NERs (NERs A and B), while task T3 also produces another two NERs (NERs C and D). The top right of Figure 2 shows the situation when these three tasks are removed from the FPGA and three new NERs (NERs E, F, and G) become available. Let's assume task T4 arrives and has to be placed. At this point, there is no single NER available that can fit this new task. In this case, IM needs to merge NERs and

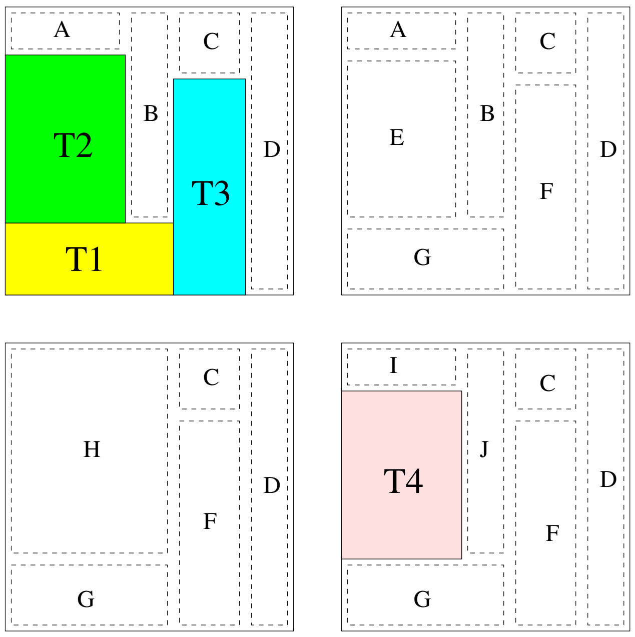

Figure 2. PM technique

form a bigger NER for this new task. In order to accommodate task T4 (bottom right of Figure 2), the PM technique in our IM algorithm only needs to perform one merge operation (NERs A, B, and E) and form a new bigger NER (NER H) (bottom left of Figure 2). Again, Bazargan's algorithm would perform additional merging. More precisely, the merging of NERs A, B, and E, then merging C, F, and D, and finally merging of all of them into one new bigger NER is required. In this example, Bazargan's algorithm needs three merging operations while our IM needs only one. We call this technique also *merge-on-demand* which is the key element of the proposed PM technique to reduce algorithm execution time.

3.3. DC technique

Direct Combine (DC) technique is a technique that allows IM to combine NERs directly without merging and splitting operations, thereby saving algorithm execution time. Figure 3 shows the working of the proposed DC technique.

As in the figures above, the top left of Figure 3 shows the beginning situation when a task T1 is placed on the FPGA. This leads to two NERs (NERs A and B). The top right of Figure 3 shows the FPGA after T1 has been completed. The new NER (NER C) is produced. Let's assume, Task T2 arrives. At this point, all NERs in this location are free, so it is possible to merge the NERs (NERs A, B, and C) to form a new bigger NER (NER D) as in the Bazargan algorithm. To

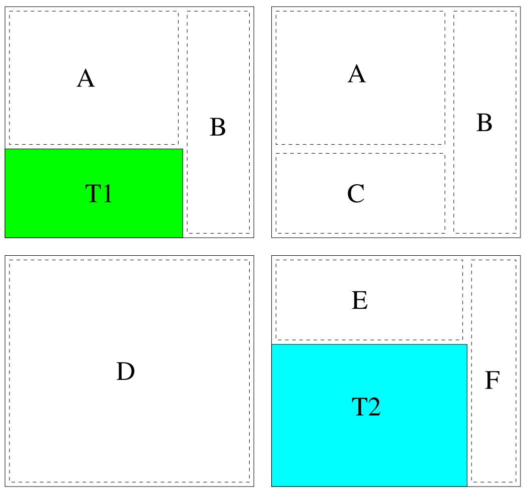

Figure 3. DC technique

decrease algorithm execution time, instead of merging (release memory) and splitting (allocate memory) NERs, the DC technique directly combines the NERs (NERs A, B, and C) to create a bigger NER (NER D) (bottom left of Figure 3). The resulting NER can be used to place the new task (bottom right of Figure 3). To increase the placement quality, the DC technique will always directly combine NERs to form a bigger NER before placing a task when possible. We call this Combine Before Placing (CBP) strategy. For example if the size of task T2 on Figure 3 is smaller than NER A, the DC technique will not directly place the task on NER A. To prevent fragmentation, our DC technique will combine these three empty NERs (NERs A, B, and C) before placing the task on this new combination NER. Therefore this CBP strategy decreases the fragmentation of empty areas and increases the placement quality.

3.4. IM algorithm

To speedup the execution time of Bazargan's algorithm without loosing its good placement quality, we propose to dynamically combine the above three techniques for small to medium task sizes. If the task is too large, the possibility that the task can be placed without merging decreases, so in this case our techniques will not work. Therefore IM will activate the techniques depending on the task sizes.

If the task is not too large, IM will do CBP or MON. If IM fails to find placement after doing CBP or MON, IM will do PM. If IM also fails to find placement after doing PM, IM will reject the task. If IM can find placement using CBP, IM will place the task using DC placement. If IM can find placement using MON, IM will place the task using normal placement.

If the task is too large, IM will do total merging before finding placement (just like the original Bazargan). If IM can find placement after total merging, IM will place the task using normal placement, otherwise IM will reject the task.

4. Evaluation

4.1. Experimental setup

We have constructed a discrete-time simulation framework in ANSI-C to evaluate the performance of the proposed techniques and algorithm and compare it to related art. Our measurements have been conducted on a Pentium-IV 3.4 GHz PC. Each task is placed at its arrival time and if the placement fails, it is assumed rejected; it is no-queue scheduling. A single new task arrives at each time unit. Furthermore, our scheduling scheme is non-preemptive – once a task is loaded onto the device it runs to completion.

We model an FPGA with size of 100x100 reconfigurable units and use tasks with randomly generated sizes and life-times. To our best knowledge, there are no standard benchmarks available to evaluate online placement algorithm. We therefore generate our own synthetic benchmark sets. To represent real-life scenarios, we generate randomly 13 task sets as depicted in Table 1 ranging from short life-time tasks (50 time units) till long life-time tasks (200 time units) and also from small size tasks (4 reconfigurable units) till large size tasks (400 reconfigurable units). The last task set (MTS) is mixed task set of TS1 to TS12.

Wmin, Wmax, Hmin, Hmax, Ltmin, and Ltmax denote minimum task width, maximum task width , minimum task height, maximum task height, minimum life-time, maximum life-time, respectively. Every task set consists of 1000 tasks assuming uniformly distributed life-times and task sizes.

Using this simulation framework, we compared our algorithm with Bazargan's algorithm. For Bazargan's algorithm, we use the First Fit (FF) heuristic for choosing NERs and Shorter Segment (SSEG) heuristic for splitting decision, because these heuristics have the best performance, as mentioned in [2].

Our study is based on two performance parameters, those are the average percentage of accepted tasks (%) and the average algorithm execution time(μs). The average percentage of accepted tasks represents the placement quality, while the average algorithm execution time is a metric for the algorithm performance. The average value is the result of 1000 iterations of the algorithm for every task set.

To study the impact of the different techniques proposed in this paper, we performed experiments with five different cases:

- BFFSSEG: Bazargan's algorithm using FF and SSEG heuristics [2];
- MON: algorithm using MON technique;
- MON+PM: algorithm using combination of MON and PM techniques;
- MON+PM+DC: algorithm using combination of MON, PM, and DC techniques;
- IM: our Intelligent Merging algorithm.

Table 1. Task sets for simulation

Task Set	Wmin	Wmax	Hmin	Hmax	Ltmin	Ltmax
TS1	2	5	2	5	50	100
TS2	2	5	2	5	100	150
TS3	2	5	2	5	150	200
TS4	5	10	5	10	50	100
TS5	5	10	5	10	100	150
TS6	5	10	5	10	150	200
TS7	10	15	10	15	50	100
TS8	10	15	10	15	100	150
TS9	10	15	10	15	150	200
TS10	15	20	15	20	50	100
TS11	15	20	15	20	100	150
TS12	15	20	15	20	150	200
MTS	2	20	2	20	50	200

4.2. Experimental result

The average percentage of accepted tasks for each task set is depicted in Figure 4. The effect of each technique on the number of accepted tasks is depicted in Figure 5. A positive value means the technique increases the number of accepted tasks, while the negative value means the technique decreases the number of accepted tasks. The average algorithm execution time over 1000 runs for each task set is depicted in Figure 6. The effect of each technique on algorithm execution time is shown in Figure 7.

4.3. Effect of task size and life-time

As the task size increases, the average percentage of accepted tasks decreases because it is more difficult to find available free space that can accommodate the task. The longer life-time task decreases the average percentage of accepted tasks, because the task will stay longer on the FPGA. It is thus more difficult to find available free space that can accommodate the other tasks.

Large tasks negatively influence the algorithm execution time. This is to be expected, because when the task size is bigger, the possibility that the task can be placed on one of NERs or combined NERs on the FPGA without merging becomes smaller. A similar observation holds for the

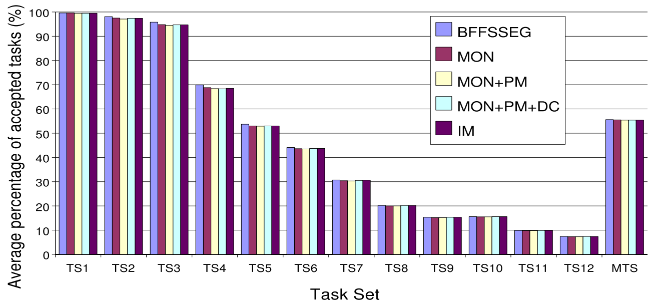

Figure 4. Average percentage of accepted tasks (%)

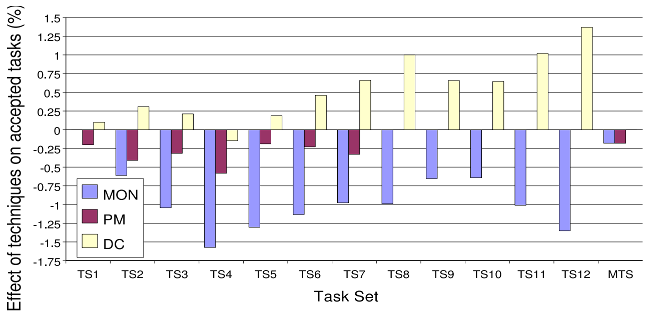

Figure 5. Effect of techniques on accepted tasks(%)

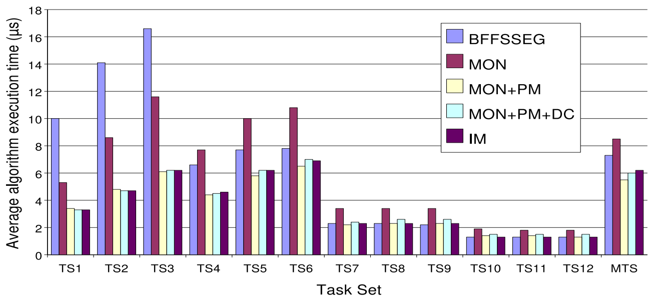

Figure 6. Average algorithm execution time(μs)

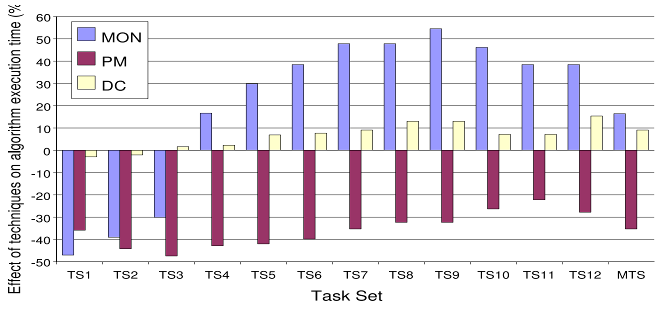

Figure 7. Effect of techniques on algorithm execution time(%)

life-time of tasks where the execution time is negatively influenced as tasks will stay longer on the FPGA. Therefore the probability that next tasks can be placed on one of the NERs without merging becomes smaller.

4.4. Evaluation of algorithm using MON technique

The algorithm using MON technique is up to 1.9 times faster than the Bazargan's algorithm with similar accepted task percentage as the result of intelligently avoiding total merging. On the average, the number of accepted tasks is reduced by 0.95 %. However for mixed task set, this is only 0.18 %.

4.5. Evaluation of algorithm using combination of MON and PM techniques

Among these algorithms, the algorithm using combination of MON and PM techniques (MON+PM) performs the best in terms of algorithm execution time on average. The algorithm is up to 2.9 times faster than the Bazargan's algorithm with similar accepted tasks as the result of intelligently avoiding total merging and exploiting its merge-on-demand capability. On the average, the decreasing of accepted tasks is 1.24 %. However for mixed task set, the decreasing is only 0.36 %.

4.6. Evaluation of algorithm using combination of MON, PM, and DC techniques

The algorithm using combination of MON, PM, and DC techniques (MON+PM+DC) is up to 3 times faster than the Bazargan's algorithm with similar accepted tasks as the result of intelligently avoiding total merging and exploiting its merge-on-demand and direct combine capability. On the average, the decreasing of accepted tasks is 0.95 %. However for mixed task set, the decreasing is only 0.36 %.

4.7. Evaluation of IM algorithm

IM can effectively exploit the advantages of our three techniques especially when the tasks are not too large, because the possibility that the tasks can be placed without merging become large.

The IM algorithm is up to 3 times faster than the Bazargan's algorithm with similar accepted tasks by intelligently exploiting the proposed three techniques. On the average, the decreasing of accepted tasks is 0.89 %. However for mixed task set, the decreasing is only 0.36 %.

On the basis of these results, we can state that our algorithm produces comparable results as Bazargan with a slight minor difference for the worst case but similar placement quality in the best case.

4.8. Effect of MON technique

The MON technique can decrease the algorithm execution time for small task sets. When the tasks are small, the possibility that the tasks can be placed on one of NERs without merging becomes bigger, so in this case MON can prevent total merging effectively.

The MON technique decreases up to 47 % algorithm execution time by intelligently avoiding total merging. On the average, the MON technique decreases 0.95 % accepted tasks. However for mixed task set, the decreasing in only 0.18 %.

4.9. Effect of PM technique

The PM technique can effectively decrease the execution time for all task sets thanks to its merge-on-demand capability.

The PM technique decreases up to 47.4 % algorithm execution time due to its merge-on-demand capability. On the average, the PM technique decreases 0.29 % accepted tasks. However for mixed task set, the decreasing in only 0.18 %.

4.10. Effect of DC technique

The DC technique decreases algorithm execution time for small task sets, as the possibility that the tasks can be placed on one of combined NERs without merging becomes bigger, so DC technique can combine NERs effectively.

We see that the DC technique increases the number of accepted tasks for almost all task sets except TS4 as the result of its CBP strategy.

The DC technique decreases up to 2.94 % algorithm execution time by intelligently avoiding merging and splitting. On the average, the DC technique decreases 0.29 % accepted tasks. However for mixed task set, it does not affect on accepted tasks.

5. Conclusions

In this paper, we have proposed the Intelligent Merging (IM) algorithm an improved version of Bazargan in terms of execution speed. Our experiments show that our algorithm is 1.72 times faster while loosing only 0.89 % of accepted tasks on average.

Our algorithm does not yet consider I/O communication and heterogeneous FPGAs. We plan to take into account these additional constraints in our future work. We also aim at the creation of standard benchmarks representing real world applications and workloads.

6. Acknowledgment

This work is sponsored by the hArtes project (IST-035143) supported by the Sixth Framework Programme of the European Community under the thematic area "Embedded Systems".

References

[1] A. Ahmadinia, C. Bobda, and J. Teich. A dynamic scheduling and placement algorithm for reconfigurable hardware. In *Architecture of Computing Systems (ARCS)*, pages 125–139, 2004.

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

[3] J. Cui, Q. Deng, X. He, and Z. Gu. An efficient algorithm for online management of 2d area of partially reconfigurable fpgas. In *Design Automation and Test in Europe (DATE)*, pages 129–134, Apr 2007.

[4] J. Cui, Z. Gu, W. Liu, and Q. Deng. An efficient algorithm for online soft real-time task placement on reconfigurable hardware devices. In *IEEE International Symposium on Object/component/services-oriented Real-time distributed Computing (ISORC)*, pages 321–328, May 2007.

[5] M. Handa and R. Vemuri. An efficient algorithm for finding empty space for online fpga placement. In *Design Automation Conference (DAC)*, pages 960 – 965, June 2004.

[6] M. Handa and R. Vemuri. An integrated online scheduling and placement methodology. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 444–453, Aug./Sept. 2004.

[7] C. Steiger, H. Walder, M. Platzner, and L. Thiele. Online scheduling and placement of real-time tasks to partially reconfigurable devices. In *Real-Time Systems Symposium(RTSS)*, pages 224– 225, Dec 2003.

[8] J. Tabero, J. Septién, H. Mecha, and D. Mozos. A low fragmentation heuristic for task placement in 2d rtr hw management. In *Field-Programmable Logic and Applications (FPL)*, volume 3203 of *LNCS*, pages 241–250. Springer, 2004.

[9] J. Tabero, H. Wick, J. Septién, and S. Roman. A vertex-list approach to 2d hw multitasking management in rtr fpgas. In *Design of Circuits and Integrated Systems(DCIS)*, pages 545–550, November 2003.

[10] M. Tomono, M. Nakanishi, S. Yamashita, K. Nakajima, and K. Watanabe. An efficient and effective algorithm for online task placement with i/o communications in partially reconfigurable fpgas. *IEICE Trans. Fundamentals*, E89-A:3416 – 3426, December 2006.

[11] M. Tomono, M. Nakanishi, S. Yamashita, K. Nakajima, and K. Watanabe. A new approach to online fpga placement. In *Conference of Information Science and Systems (CISS)*, pages 145–150, March 2006.

[12] H. Walder, C. Steiger, and M. Platzner. Fast online task placement on fpgas: Free space partitioning and 2d-hashing. In *International Parallel and Distributed Processing Symp. (IPDPS)*, page 178, April 2003.