Speeding up Online Placement for XILINX FPGAs by Reducing Configuration Overhead

Ali Ahmadinia Jürgen Teich
Department of Computer Science 12
Hardware-Software-Co-Design
University of Erlangen-Nuremberg, Germany
{ahmadinia, teich}@cs.fau.de

Abstract

Recent generations of FPGAs allow run-time partial reconfiguration. To increase the efficacy of reconfigurable computing, multitasking on FPGAs is proposed. One of the challenging problems in multitasking systems is online template placement.

In this paper, we describe how existing algorithms work, and propose a new multi-stage method for mapping of tasks to reconfigurable hardware. Also a new fitting algorithm, as a part of online placement, has been developed. This method tries to reduce the reconfiguration overhead. First simulation results are promising.

1. Introduction

The introduction of configurable hardware logic devices in the 1980s created an alternative method for high performance in applications where neither the general purpose processor nor application specific integrated circuit (ASIC) proved to be the preferred choices. The general-purpose processor achieves acceptable results for a wide range of applications and the ASIC achieves improved performance, but for a concentrated set of applications. The logic of a configurable device can be programmed more than once to fit the user's needs at the time of execution. This hybrid device can result in better performance than the general purpose processor without the fabrication expense of the ASIC.

Reconfigurable systems can offer both high degrees of parallelism and flexibility for compute-intensive applications. In a run-time reconfigurable environment, where partial (or full) reconfiguration of the hardware device is allowed, both attributes offer advantages over typical general-purpose processors. In such a system, configuration data can be downloaded without interrupting execution of previously loaded applications. For this reason, applications must be placed as quickly as possible to service the user's need.

The improvements in logic capacity and performance have increased the usefulness of FPGAs and broadened the range of potential applications of the technology. However, these improvements have neither resolved nor eliminated issues pertaining to efficient placement of applications on configurable logic devices. Run-time reconfigurable systems require fast placement of mappings to the device(s), something that is not always easily achieved, despite increased capability.

We aim at decreasing reconfiguration overhead as much as possible, to achieve a better performance in online placement.

This paper is organized as follows. Section 2 details previous work in the area of online placement algorithms for reconfigurable hardware. Section 3 will present a new multi-stage mapping of tasks to reconfigurable hardware. In fact, we have delayed deleting of module from reconfigurable hardware. Section 4 will cover the details of our fitting strategy, which is based on column based configuration of Xilinx Virtex. The simulation results will be shown in Section 5.

2. Previous Work

For the offline version of placement, an optimal method has been developed, by using 3D placement of tasks in time and area dimension for dynamic reconfigurable hardware [1] (see figure 1).

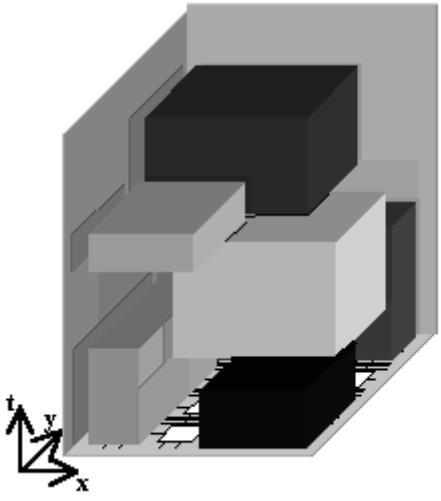


Figure 1. Offline 3-D placement

In the offline problem there is no time constraint for finding an optimal placement. But in online placement, modules are processed in the time they start and the algorithm only looks at the current cut plane when deciding on whether there is room for a new module or where to place it. There are two main parts in online placement algorithms. The first part is handling empty rectangles. Bazargan et al. [2] have implemented two categories of methods: 1. Keeping all the maximal empty rectangles (KAMER). 2. Keeping disjoint empty rectangles. These methods are shown in figures 2 and 3, respectively. Since the first method keeps all the maximal empty rectangles and hence checks all of them for placing an arriving module, the quality of its placement is better than any method the second category, provided that the same bin-packing rule is used. However, methods of the second category are faster. Walder et al. [3] presented three newly developed partitioning algorithms based on Bazargan's [2] approach. They have also suggested a hash matrix approach to maintain the free space. The second part of placement is similar to online bin-packing problem. Bazargan et al. [2] have proposed two-dimensional extensions of First Fit (FF), Best Fit (BF) and Bottom Left (BL).

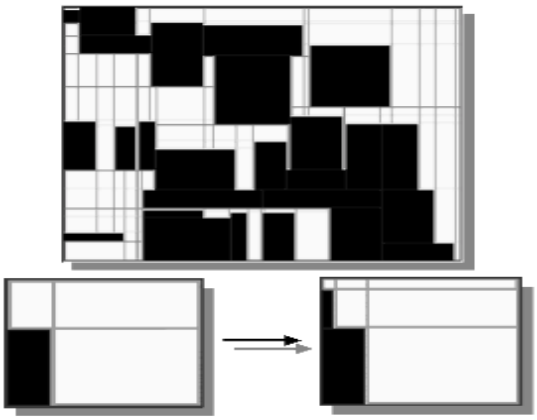


Figure 2. Keeping all maximal empty rectangles [2]

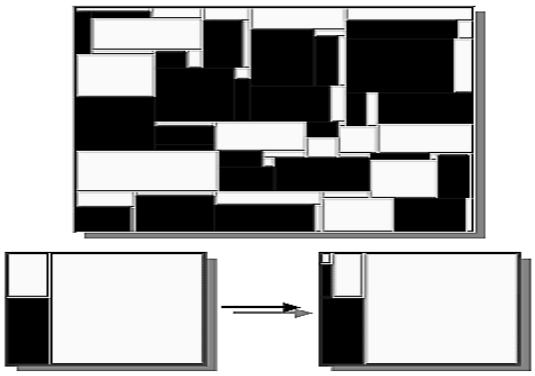


Figure 3. Keeping non-overlapping empty rectangles [2]

3. Online-Mapping of Tasks to FPGA

In contrast to related work, after finishing the task, we will a) not delete the configuration of the corresponding module from the FPGA. Because a task graph may have a periodic behavior, and the same task may be requested again, then we can map it to the preconfigured and inactive module on hardware without any configuration overhead, and no update of the free rectangle database is necessary. Furthermore, at any arriving task request, we will b) first search if the same task is running on the FPGA. Then, by computing the mobility of a task with respect to a given deadline, we will decide either to run it later on the same place, or to locate a new module. The details of this approach are illustrated in the following algorithm:

```
task_request (t);
check_same_task_on_hardware (t);
If (found_same_task (t))
{
  ALAP (t) = Deadline (t) – Execution (t);
  ASAP (t) = end_of_same_task (t);
  Mobility = ALAP (t) - ASAP (t);
If (Mobility >= 0)
{
Task "t" will be kept, until the termination of the same
task on hardware, and then will be mapped to the same
module.
}
}
```

Figure 4 shows the proposed system model, consisting of a host processor, and a partially reconfigurable hardware device. The FPGA provides reconfigurable logic elements as a 2-dimensional CLB (Configurable Logic Block) array. The placer and loader form the part of a reconfigurable operating system dealing with resource management. The reconfigurable OS is run on a host processor.

The general steps for mapping a task can be done in the following way, in each step the process will be finished if a solution is found:

I. Check if the task t can be executed on a preconfigured and inactive module.
II. Check, if the same task is running now on the FPGA, and then compute the mobility of the task (This step has been explained in details above).
III. Check online placement.
IV. Delete preconfigured and inactive modules from reconfigurable hardware, then check online placement again.

If the online placement can not find a feasible space, the task will be checked again by online placement at its *ALAP* time. This time if the online placement is not successful, the task has to be rejected.

The management for reactivating modules is very simple. For each module, we define *start*, *reset*, and *finish* signals. When we configure a module on the hardware, first its *start* input and then the *reset* input will be activated. Then by finishing the computation of the assigned task, the *finish* output will become high. Therefore when the *finish* signal of one module is high, it shows that the module is inactive and can be reactivated by using its *reset* and *start* inputs.

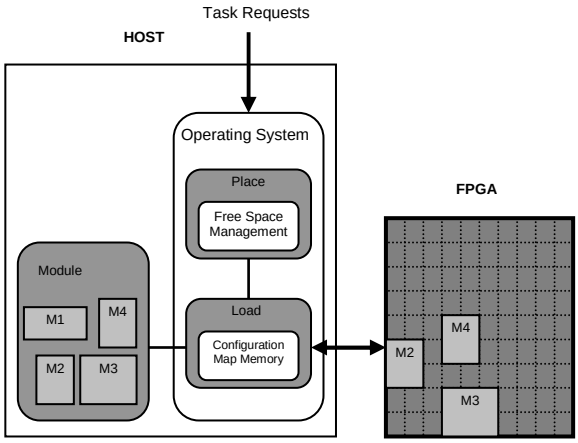


Figure 4. The structure of reconfigurable operating system

4. Online Placement

4.1. Partitioner

Our generic online placement algorithm consists of two main parts. First, an empty space partitioning manager both for insertion and deletion of modules. Second, a fitter for choosing a free rectangle for placing a module.

As mentioned before, different methods of partitioning have been developed in related work. Here, we apply the KAMER method [2]. Although it has not a good performance, it has the best quality. But this is not important because we want to evaluate our fitting strategy against other fitting methods in the following.

4.2. Fitter

A fitter decides which empty rectangle to be used for placing a module. We suggest a fitting algorithm to reduce the reconfiguration overhead, because reconfiguration overhead is a main factor that prevents dynamic reconfiguration from being accepted. With the advance of FPGA technology, the reconfiguration time is reduced dramatically. However, compared with ever increasing FPGA speed, the reconfiguration delay is still quite considerable. The configuration time often is 16% to 71% of total running time [2,5].

Currently available FPGA technology (Xilinx Virtex) can be partially reconfigured since frames can be read or written individually. Note that it is not possible to configure a single CLB, since the frames belonging to a given CLB are common to all other CLBs in the same column. So, if a modification to a single CLB is required, all frames belonging to the same column must be read (operation called read-back), and the modification is inserted over the read frames. Hence, the configuration of a task potentially interferes with other tasks allocated to the same columns [6,7,8]. Then, the interfered tasks will be stopped, during the configuration of the new module. Also, there is an extra time overhead for state extraction and task reconstruction [9], to suspend the interfered tasks and to restore them.

Therefore, we propose a method with the criteria of interfering running modules as least as possible. We name this method "Least Interference Fit (LIF)", which chooses the least low possible rectangle. The lowness of a rectangle is defined as follows:

***Lowness of an empty rectangle**:*

Maximum number of modules below the rectangle in every column of the rectangle.

Previous fitting algorithms are as follows:

- The FF algorithm puts the arriving task in the lowest indexed empty rectangle that can accommodate the task.
- The BF algorithm chooses the smallest free rectangle which can accommodate the task.
- The BL strategy tries to place a task in a sufficiently big free area whose lower left corner is located closest to the lower left corner of the FPGA.

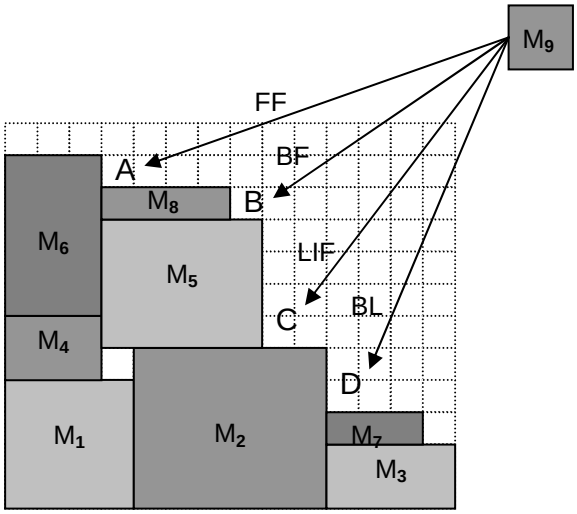


Figure 5. Comparison of LIF approach with other methods

Let us use an example to show how our fitting algorithm works. Figure 5 shows that we want to place module M_9 on the FPGA. The First Fit (FF) method chooses position A, then for placing the new module, we must interfere with four modules: M_1, M_2, M_5 and M_8. The Best Fit (BF) algorithm puts the module on position B. In this case, M_2 and M_5 will be interfered. The Bottom Left (BL) strategy places module M_9 on position D, but here we have to stop two modules namely M_7, M_3 too. In our approach (LIF), we put the module at position C that will interfere only with module M_2.

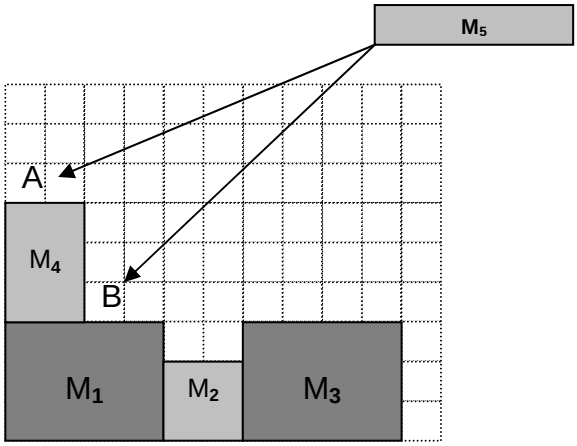


Figure 6. Example of normal fitting without rotating the module

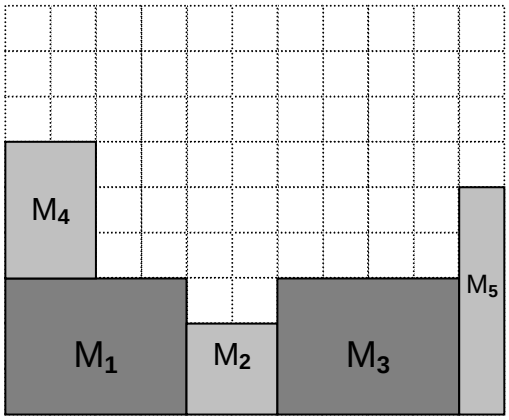


Figure 7. Placing module M5, after rotating the module

As a second possibility to decrease the interference of other modules, we may change the shape of modules before placement. We assume that modules are relocatable, and as mentioned in [10], it is possible to change the routing programmed into each cell to reflect the overall rotation of the configuration. Therefore, before placing a module M_i, with width of W_i and height of H_i, we check if H_i is smaller than W_i, if yes, we rotate the module.

In Figure 6, we have shown the problem if we don't use this rotation. For placing M_5, we have two choices: position A and position B. If we want to place this module in position A, modules M_1, M_4 and M_5 must be stalled, during the configuration time. In the case of using position B, modules M_1, M_2 and M_3 will be interfered.

Before starting the placement process, if we rotate module M_5, we can place M_5 without interfering any other running modules as illustrated in Figure 7.

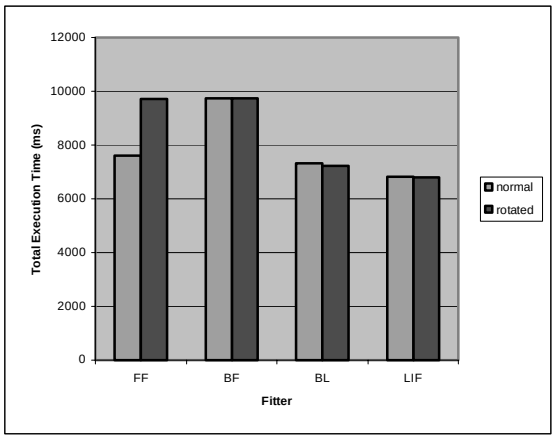
5. Experimental Results

To investigate the influence of our proposed method for Fitter, we have implemented a system model, with randomly generated task sets. For the size of the tasks, we have used two different classes. Classes 1 and 2 contain tasks with uniformly distributed size of module dimensions in the ranges 5 to 35 and 5 to 50, respectively.

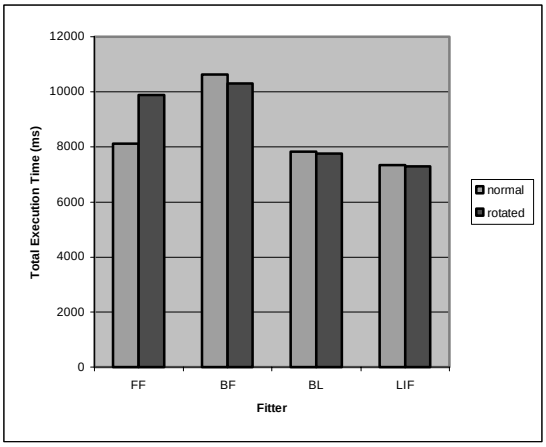
In our simulation framework, the tasks are independent, and we assume that tasks cannot be preempted. Each task set consists of 50 tasks, and the arrival time of tasks is distributed in 0 to 3 time units, to eliminate the influence of late arriving tasks on the overall execution time [7].

The size of chip has been assumed 56x84 and 80x120, a 2-dimensional CLB array, corresponding to the Xilinx Virtex XCV800 and XCV2000E devices respectively. The running time of tasks is distributed uniformly in the interval of 20 to 200 time units.

Figure 8 shows the results for chip size 56x84. As depicted in Figure 8, the LIF method has the best performance, and BL is also obtaining small total execution time. When allowing rotation, we can see that FF becomes worse, and other fitters show only a little improvement. In a second experiment, we increased the chip size to 80x120. As shown in Figure 9, for LIF and BL, the execution time decreases. In this case, BF (like FF) produces longer execution time.

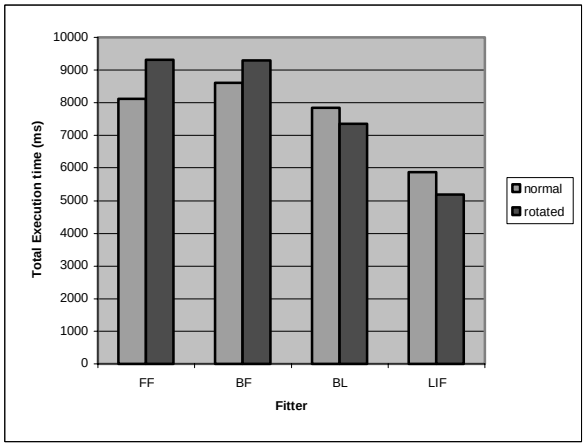


(a)



(b)

Figure 8. Performance of fitting algorithms (Chip Size: 56x84) (a) class1 (b) class2



(b)

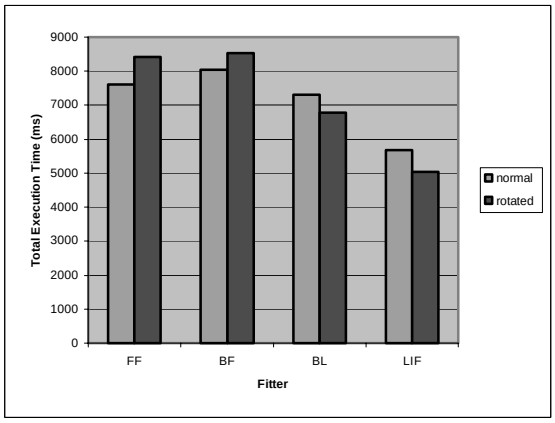
Figure 9. Performance of fitting algorithms (Chip Size: 80x120) (a) class1 (b) class2

6. Conclusion

In this paper we have discussed online placement techniques for reconfigurable FPGA. We suggested a multi-stage mapping of tasks to modules, and a new fitting strategy for online placement. Also, we have extended our fitting algorithm to include a pre-rotation.

We have conducted experiments to evaluate FF, BF, BL placements, and also our LIF method. In all cases, LIF method has the best performance. Using rotation for FF and BF gives sometimes even worse results, because rotation makes modules higher, and that is not proper for FF and BF. But for BL and LIF, rotation always improves the total execution time, especially for larger chip sizes.

Concerning future work, we plan to evaluate our multi-stage mapping of tasks to reconfigurable hardware with realistic task sets. We also intend to develop an efficient scheduling algorithm for meeting task deadlines.



(a)

7. References

[1] Sándor Fekete, Ekkehard Köhler, and Jürgen Teich, "Optimal FPGA Module Placement with Temporal Precedence Constraints", In *Proc. of Design Automation and Test in Europe* , IEEE-CS Press, Munich Germany, 2001, pp. 658-665.

[2] Kiarash Bazargan, Ryan Kastner, and Majid Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems", In *IEEE Design and Test of Computers*, volume 17, pages 68-83, 2000.

[3] Herbert Walder, Christoph Steiger, and Marco Platzner, "Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing", In *Proc. of International Parallel and Distributed Processing Symposium (IPDPS) / Reconfigurable Architectures Workshop (RAW).* IEEE-CS Press, Nice France, April 2003.

[4] Grant Wigley, and David Kearney, "Research Issues in Operating Systems for Reconfigurable Computing", In *Proceedings of the 2nd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA).* CSREA Press, Las Vegas USA, June 2002.

[5] M. J. Wirthlin, and B. L. Hutchings, "A dynamical instruction set computer", In Proc. *of Field-Programmable Custom Computing Machines*, 1995, pp. 99-107.

[6] Daniel Mesquita, Fernando Moraes, Jos'e Palma. Leonardo Möller, and Ney Calazans, "Remote and Partial Reconfiguration of FPGA: tools and trends", In *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS) / Reconfigurable Architectures Workshop (RAW).* IEEE-CS Press, Nice France, April 2003.

[7] Herbert Walder, and Marco Platzner, "Online Scheduling for Block-partitioned Reconfigurable Devices", In *Proc. of Design Automation and Test in Europe* , IEEE Computer Society Press, Munich Germany, 2003, pp. 290-295.

[8] "Virtex FPGA Series Configuration Architecture User Guide", *XILINX XAPP151* v1.5, September 2000.

[9] L. Levinson, R. Männer, M. Sessler, and H. Simmler, "Preemptive Multitasking on FPGAs", In *Proc. Symp. Field-Programmable Custom Computing Machines (FCCM'00)*, Napa, CA, USA, 2000, pp. 301-302.

[10] Katherine Compton, James Cooley, Stephen Knol, and Scott Hauck, "Configuration Relocation and Defragmentation for Reconfigurable Computing", In *Proc. of FPGAs for Custom Computing Machines (FCCM)*. IEEE-CS Press, April 2001.