Task Scheduling for Heterogeneous Reconfigurable Computers

Ali Ahmadinia, Christophe Bobda, Dirk Koch, Mateusz Majer, Jürgen Teich
Department of Computer Science 12, University of Erlangen-Nuremberg, Germany
{ahmadinia, bobda, dirk.koch, mateusz, teich}@cs.fau.de

ABSTRACT

We consider the problem of executing a dynamically changing set of tasks on a reconfigurable system, made upon a processor and a reconfigurable device. Task execution on such a platform is managed by a scheduler that can allocate tasks either to the processor or to the reconfigurable device. The scheduler can be seen as part of an operating system running on the software or as core in the reconfigurable device. For each tasks to be executed on reconfigurable device, an equivalent implementation exists as rectangular block in a database. This block has to be placed on the device at run-time. A placer is responsible for the placement of tasks received from the scheduler on the reconfigurable device. However, the placement of tasks on the reconfigurable device will not be succesful if enough space is not available on the device to hold the task. In this case the scheduler receive an acknowledgment from the placer and decide either to preempt a running task or to run the task on software. We present in this work an implementation of a placer module as well as investigations on task preemption. The two modules are part of an operating system for reconfigurable system currently under development.

Categories and Subject Descriptors

J.7 [**COMPUTER-AIDED ENGINEERING**]; C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]; C.4 [**PERFORMANCE OF SYSTEMS**]: Design Studies

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

FPGA, Reconfigurable Computing, Partial Reconfiguration, Scheduling, Placement, Hardware Preemption

1. INTRODUCTION

A reconfigurable computing system is usually composed of a host processor and a reconfigurable device such as an SRAM-based Field-Programmable Gate Array (FPGA) [9]. Depending on their characteristics, tasks can be allocated either to the processor or to the reconfigurable device for execution. Most of the time, the processor is used for sequential tasks (software tasks), while the reconfigurable device is used for acceleration of tasks with high degree of parallelism (hardware tasks). Meanwhile tasks exist which can be executed on software as well as on hardware (hybrid tasks).
The scheduling of software tasks on a processor can be done by well known methods existing in the literature [5]. However, with the ability of partial reconfiguration and multiple tasks which can be configured separately and executed simultaneously on a reconfigurable device, the problem become more complex, since multitasking and partial reconfiguration increases the device utilization. It also necessitates well thought dynamic task placement and scheduling algorithms. Such algorithms strive to use the device area as efficiently as possible as well as reduce total task configuration and running time.
These methods can be embedded in an abstraction layer in host processor, which this layer is called operating system [10].
The execution of a task on the reconfigurable device leads to the online placement problem, for which a method based on free rectangles management and heuristics fitting has been proposed in [3] and improved in [9][2].
Online scheduling on hardware is very dependent on placement, which restricts the operating system to schedule tasks optimally. In most of existing work on scheduling of tasks in reconfigurable computers, it is assumed that all tasks should be executed on hardware [8]. Moreover hardware tasks are not allowed to be preempted. We present a novel on line placement method as well as scheduling techniques which also use preemption of hardware tasks to minimize task rejection.
In the next section, the details of an operating system for reconfigurable computers will be explained. Section 3 presents our online placement algorithm used by the placer to allocate space on the chip to incoming tasks will provide information of online placement, and section 4 consists of our main contribution for scheduling. In section 4.3 we investigate the preemption of hardware tasks and propose two methods to solve this problem. Evaluation of the proposed methods will be presented in section 5, then the work will be concluded in section 6.

2. BACKGROUND

Before explaining the online placement and scheduling problem, the definitions and assumptions of our system model according to the related work in this area will be given. We assume that a reconfigurable device R is made upon a set of reconfigurable Processing Elements (PE) arranged in rectangular array with H rows and W columns. The PEs can be somehow connected together.

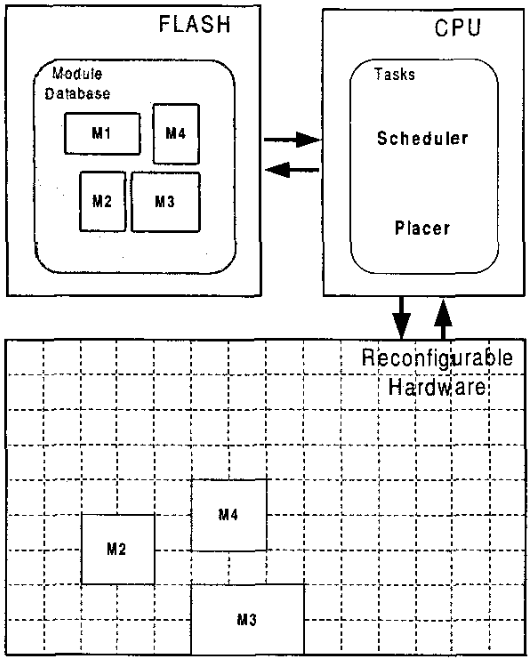

Figure 1: A multitasking system on reconfigurable device

Our model of dynamic reconfiguration is made upon a scheduler, an online placer and the reconfigurable device (Figure 1). The scheduler manages the tasks and decides when and on which device (hardware or software) a task should be executed.
For tasks to be executed in hardware, the scheduler requests a free space on the reconfigurable device. Therefore it addresses a request to the placer which tries to find and assign a proper free space. If the placer fails to find a free space, it acknowledges the scheduler which then decides either to run the task in software or to try it again later on hardware or to reject it. If the placement is successful, then the placer will configure the reconfigurable system and acknowledge the scheduler. In Figure 2, the main steps of this system are shown.
For each task to be placed on the reconfigurable device, we further assume that an implementation as a rectangular block is available with the input and output ports on its boundary. This implementation is stored in a module database, and will be retrieved on demand. We define the characteristics of a task as follows:

DEFINITION 1 (TASK CHARACTERISTICS). *Given a set of tasks $T = \{t_1, t_2, ..., t_m\}$, and a task $t_k \in T$ is the tupel (a_k,e_{swk},e_{hwk},e_{cfk},w_k, h_k,d_k) where*

- *a_k is arrival time of task t_k*
- *e_{swk} is execution time of task t_k on software*
- *e_{hwk} is execution time of task t_k on hardware*
- *e_{cfk} is configuration time of task t_k on hardware*
- *w_k is width of the corresponding module for task t_k in terms of PE*
- *h_k is height of the corresponding module for task t_k in terms of PE*
- *d_k is deadline time of task t_k*

3. TASK PLACEMENT

The first part of the placement problem is to identify all possible sites where a new module can be placed.
Bazargan et al. [3] proposed to store only the free spaces as maximal rectangles, however the complexity of this approach is $O(n^2)$ where n is the number of modules running on the device. However, in Bazargan's approach, a binary tree is used for managing empty rectangles, which is complicated to keep updated by deletion and insertion of modules, because in some cases many nodes of the tree have to be changed. Hence we propose a simpler and faster approach with complexity of $O(n)$. Contrary to the Bazargan's approach, we propose to manage the occupied space of the device rather than the free space. Our approach is based on the observation that the set of empty rectangles on the device grows much faster than the set of placed components, therefore it will be much faster to use the occupied space to find the set of free sites where the new component can be placed.
Without loss of generality, we will consider that components are

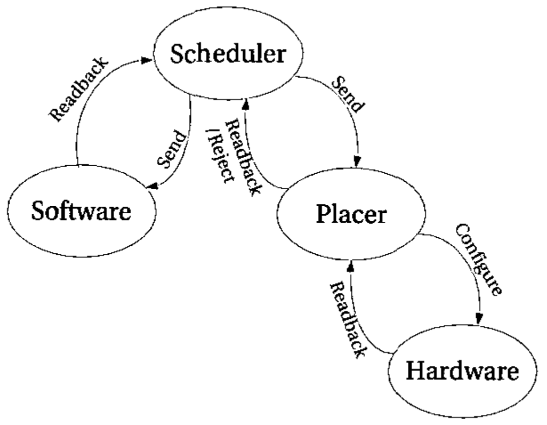

Figure 2: Main states of the multitasking system

placed relatively to their lower left positions. We first define the Impossible Placement Region (IPR) for a new component.
For a new component c to be placed on the device and a placed component c', the Impossible Placement Region (IPR) $I_{c'}(c)$ of c relative to c' is the region where c cannot be placed without overlapping with c'.
The Impossible Placement Region (IPR) $I_{c'}(c)$ of c relative to c' is identified by computing the left margin with size $h_c - 1$ and the bottom margin with size $w_c - 1$ of the component c' as illustrated in figure 3. The Possible Placement Region for the component c is obtained by subtracting the IPR from the device area.
Computing the IPR relative to the device and each placed component as described before gives us the set of IPRs as shown in figure 4. The solution of subproblem 1 can be obtained by set subtracting the IPRs from the total device area.
Since we have to compute the extended margins for the running modules and two regions relative to the device for solving the first subproblem, the required steps of our algorithm is $O(n)$.
The second part in online placement is to find the optimal position

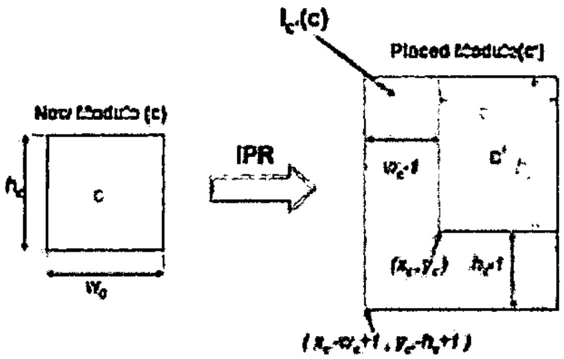

Figure 3: IPR of a new module relative to a placed one

for placing the new module from the set of possible placement regions. Our main goal here is to place the components in such a way that the communication among components is optimal. This goal can be reached by placing connected components nearby each other. Furthermore, components which have off-chip connections should also be placed on the boundary of the device, not far from the pins that they use. We will first compute the point p_{opt} which gives us the optimal placement cost. If p_{opt} is located within the Possible Placement Region (PPR) then we have the solution of second subproblem. Otherwise, we will look for the closest point to p_{opt} which is located in the PPR and select it as the optimal placement position. We define the cost to minimize as the communi-

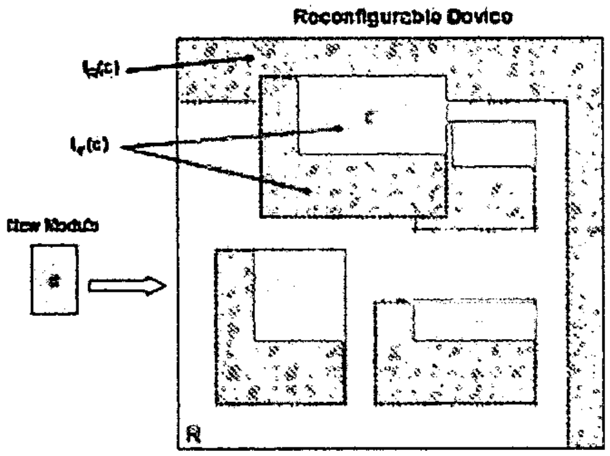

Figure 4: The area of impossible positions for the new module

cation cost among the components placed on the device in terms of distance and buswidth. We call this cost the routing cost and formally define it as follows:

Definition 2 (Routing Cost). *For two modules i and j, we define the routing cost between modules i and j as follows:*

$$RoutingCost(R_{ij}) =$$
$$((x_j + \frac{w_j}{2} - x_i - \frac{w_i}{2})^2 + (y_j + \frac{h_j}{2} - y_i - \frac{h_i}{2})^2) \times w_{ij}$$

Here w_{ij} is the width of bus connecting the two elements i and j. For finding the optimal point, we minimize the routing cost of the new module to all other placed ones. The details of this process is presented in [1].

4. TASK SCHEDULING

As we explained before, in our system model, the scheduler should decide for each task to be executed on software or hardware. Also to schedule the sequence of tasks for hardware and/or software to minimize task rejection and hardware-software communication and finally to maximize resource utilization and performance.
As it is explained in introduction, there are three types of tasks: software, hardware and hybrid tasks. The software tasks will be sent to the software queue and hardware tasks to the hardware queue. About hybrid tasks, it will be checked if the hybrid task is feasible to run on hardware and/or software, then in according to its feasibility, it will be assigned to hardware or software or both. Second part of scheduler manages the queues of tasks for hardware and software parts, where those two queues may have some common tasks. In addition, we cache modules on hardware to reduce configuration overhead. Before explaining the scheduling algorithm, the details of module caching will be given.

4.1 Module caching

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. 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 only case that we try to delete modules from hardware is that if some tasks are queued, and the queued module can not be placed on reconfigurable device because of insufficient free area. Then the inactive and pre-configured modules which are not assigned for another queued task will be removed in order of Least Recently Used (LRU) until the new module can be accommodated.
For management of reactivating modules, we define start, preload and finish signals for each module. Then by preloading the new values to the inactive module, the module can be reactivated by start signal.

4.2 Algorithm

In the first part, as mentioned before, the feasibility of hybrid tasks in hardware and software will be determined. When hw_i will be one, it means the task is feasible to be scheduled on hardware, otherwise it will be zero, and in the similar way for sw_i in software($finish(m_k)$ is finishing time of assigned tasks to module m_k) :

```
Task-Request($t_i$)
if(the same task as $t_i$ is on hardware : module $m_k$ )
  if($d_i - e_{hwi} > finish(m_k)$)
  begin
    $hw_i = 1$
    $config_i = 0$
    $s_i = finish(m_k)$
    $finish(m_k) = finish(m_k) + e_{hwi}$
  end
elsif($d_i - e_{hwi} - e_{cfi} > a_i$)
begin
  $hw_i = 1$
  $config_i = 1$
end
else
  $hw_i = 0$
if($d_i - e_{swi} > a_i$)
  $sw_i = 1$
else
  $sw_i = 0$
```

Now according to the its feasiblity, the task will be send to hardware or software queue :

```
if ($hw_i = 1$ and $sw_i = 0$)
  send $t_i$ to hardware queue
elsif ($sw_i = 1$ and $hw_i = 0$)
  send $t_i$ to software queue
elsif ($hw_i = 1$ and $sw_i = 1$)
  send $t_i$ to both software and hardware queues
else
  reject $t_i$
```

In the second part, the task queues of hardware and software must be scheduled, also we should consider these two queues may have common tasks. According to Horn Theorem [5], Earliest Deadline First (EDF) Scheduling is optimal with respect to minimizing the maximum lateness, which we try to use in our model. Therefore we sort the tasks according to their deadlines in the queues. When a task is the first one in the queue, its feasibility must be checked again, because of waiting time. If it is not feasible then it will be rejected. If it is a common task between the queues, only then it will be removed from the related queue. In software queue, the task will be started to execute after this feasibility check, but in hardware it should be checked if there is enough free area on hardware to accommodate it, and if it is failed, then the hardware caching will be used. If the caching also does not help, we will preempt the running tasks according to their lateness of their deadlines, until a feasible area for the new module is found. Each preempted task will be scheduled again in hardware, and maybe software queue. The context of preempted modules can be switched between hardware and software. This issue will be explained in details in the next section. Tasks which are assigned to a running module on hardware without any configuration overhead, will not be placed in these queues. These tasks are sorted with respect to their starting times (s_i), and by reaching the real-time to each of these starting times, they will be started on hardware.

Software Queue

```
Tasks are sorted according to their deadlines.
$t_i$ =first task from the software queue
if($t > d_i - e_{swi}$)
begin
  if($hwi = 0$)
    reject $t_i$
  else
    $sw_i = 0$
end
else
  execute $t_i$ on software
```

Hardware Queue

```
Tasks are sorted according to their deadlines.
$t_i$ =first task from the Hardware queue
if($t > d_i - e_{hwi} - e_{cfi}$)
begin
  if($swi = 0$)
    reject $t_i$
  else
    $hw_i = 0$
end
else
begin
  accept=place $t_i$ on hardware
  if(accept=0)
    accept=caching($t_i$)
  if(accept=0)
    accept=preempt(running task with latest deadline)
  if(accept=0)
    reject $t_i$ end
```

4.3 Task Preemptive Challenges

In order to achieve real reconfigurable computing it is obligatory to allow hardware tasks to be preempted. This allows us to continue a module's work at a later point in time. On preemption, the context of a task (e. g. the set of registers used by the task) should be saved. Therefore, we have to introduce some mechanisms to freeze and read out the register state in the task preemption phase. On resumption, we have to restore the context of the suspended module In contrast to the software world, this is unlikely harder to implement in hardware. The register state of a hardware module can contain a magnitude of memory bits more as compared to a software task running on a CPU. As opposed to hardware tasks, processors have dedicated commands (PUSH, POP, MOV, etc.) to (re-)store theirs context.

The majority of today's commercial available FPGAs are not fitted with extra capabilities for a fast switching of the register state. In [4] the XC6200 was used as a platform for reconfigurable computing. As all reconfiguration data of this architecture is accessible randomly, it is suitable for partial reconfiguration as well as for hardware task preemption. However, Xilinx stopped this product line.

Before focusing on the different methodologies to readback the state of a hardware task we reveal common issues on hardware preemption.

Similar to a typical processor that starts on an interrupt by finishing the just running instruction before performing the jump to the handler routine, a running hardware module could also require dedicated points of time where the module can be frozen. Multi cycle I/O transfers (e. g. DRAM access, serial links) are not allowed to be interrupted without extra precautions. In addition, multiple clock domains and clock generation units (e. g. dividers, PLLs, DLLs) have a clock dependent state that needs to be restored for a proper restart of the module. Finally, asynchronous modifications of flip-flops, uncontrollable clocks and combinatorial feedback loops make the realization of hardware task preemption on a reconfigurable device challenging. In the following we present two methodologies for hardware preemption. The first one is based on reading back the configuration bitstream while the second one is a scanpath based method.

4.3.1 Readback Methodology

Some FPGAs allow us to read back the reconfiguration bitstream. This capability was used in [7] to capture the register state. The main advantage of this methodology is that it is easy to implement. For modular placement techniques, it is essential to readback partial bitstreams without influencing other running modules. The data captured by reading back the bitstream contains the complete configuration context and not only the register state of the module. In [7] only the bits representing the register state were filtered out from the bitstream in order to efficiently store the register state. This requires a documented bitstream format or the use of dedi-

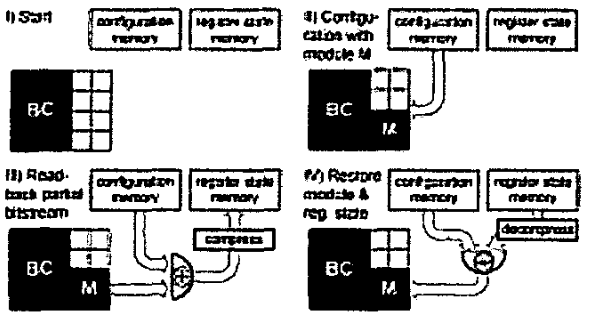

Figure 5: Capturing and restoring of a register state using the readback methodology.

cated tools. In contrast to this work, we propose a general approach as shown in figure 5 Here, the state of a module *M* is efficiently stored by the use of a compressor block. In order to gain significantly the compression ratio, we propose a bitwise XORing of the original module configuration stream together with the readback bitstream. Information equal in both streams (e. g. routing information) will lead to zeros inside the stream. The resulting stream will only contain a '1' whenever they differ (the state has changed). As the amount of data representing a register state inside the configuration bitstream is quite low [6], the majority of the resulting bits will be '0'. This approach demands some assumptions. First of all, the readback bitsteam must contain exactly the same length as compared to the configuration bitstream. In addition, every portion of data located inside the streams has always to correspondent to same reconfigurable resource on the reconfigurable architecture. Finally, ciphered bitstreams are not allowed. A small change of the state can lead to completely scrambled data in a ciphered reconfiguration context. As a consequence, the XORing will not decrease the entropy in this case.
The drawback of this approach is that the register state capturing process is done by reading back the complete bitstream. This consumes about the same time as compared to the relatively slow reconfiguration process.

4.3.2 Scanpath Methodology

One solution to decrease the time consumption of the capturing process is to include a register scanpath into the design. This methodology is close related to design for test (DFT) techniques. Including a scanpath will decrease the performance (maximum clock frequency) and increase the hardware consumption. The effects of these influences can be minimized by reducing the amount of memory inside a hardware module (e.g. reducing pipeline stages). Beside the fast register state capturing process this solution is platform independent. It is applicable to architectures not fitted with a readback capability.
Figure 6 reveals an example of how a scanpath can be included into the design of a hardware module. The scanpath should follow the structure given by the datapath of the module. This will spare routing resources on the one hand and will minimize the negative influences of the scanpath on the other. Figure 6 discloses that a global signal has to be distributed for switching the registers between scanpath and normal operation mode. This can be critical on architectures that allow no appropriate clock control. In this case the signal has to be distributed in a single clock cycle.

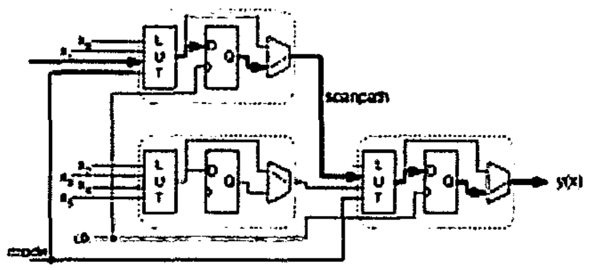

Figure 6: Capturing and restoring of a register state using the scanpath methodology.

Some modules may contain resources that can not be included directly in the scanpath register chain (e. g. memory blocks). In this case we have to build wrappers around this resources. Such wrappers contain small state machines that support capturing as well as restoring the context of these resource blocks.

4.3.3 Hardware-Software State Migration

One outstanding aspect for hardware-software-systems is the potentiality to migrate the state of a module from or to any alternative of software or hardware. Let us assume to have a hardware module synthesized for different hardware architectures by the use of a structural specification. Then it may be possible to use the same context format to restart the module on the various hardware architectures if we use the scanpath methodology to capture the register state. This will only work out if the internal structures of the reconfigurable architectures lead not to completely different hardware realizations. In other cases, context transformers can be used to arrange the register context in an arbitrary fashion. Such context transformations may further allow switching a module's work from hardware to software or vice versa. A proposal for a system that allows random choices of how the state should be executed (hardware or software) after a task has been preempted is presented in figure 7.

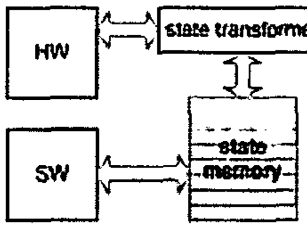

Figure 7: Register state migration by the use of a state transformer.

A digital filter for example can be implemented to use the same data types and operations in software as well as in hardware. In order to transform the context of the hardware module to a software module, the data has to be composed in the right order. This may need some additional work. For example, shift operations on FPGAs can be mapped into hardware just by using the interconnect network while in software an extra shift operation has to be executed.

5. EXPERIMENTAL RESULTS

We have implemented a system model, with randomly generated task sets. This was done for different task sizes and shapes, precisely for tasks with width and height uniformly distributed in different intervals. To see the behavior of algorithm with very different sizes of tasks, intervals of [20,40], and [20,30], and also to observe the behavior for nearly square shaped tasks [25,30] range (Table 1) has been used. The size of chip has been assumed 56x84 and 80x120, a 2-dimensional PE array, similar to the Xilinx Virtex XCV800 and XCV2000E FPGA devices, respectively.
As depicted in the following table, our method is compared with First Fit (FF) and Best Fit (BF) methods. Our method gives the least costs and Best Fit gives the most routing cost. The First Fit method in the case of nearly square tasks reaches a comparable result to our method, but in all cases needs more costs for routing. On the other side, First Fit behaves much better than the Best Fit in communication costs. Also when we use a larger chip size, our fitter works better than First Fit and Best Fit, because it has more space to find an optimal position.
For evaluating our scheduling algorithm we have used the same task size distribution for chip size 80x120. According to [6], FPGA implementation is often 10 times faster than the processor version for the same task. Thus we assume hardware execution time is distributed in range of 5 to 50 time units, and for software in range of 50 to 500 time units. The arrival times in [0,50] time interval, and deadline times in the range [0,100] with the offset of arrival and execution time is generated. Table 2 shows the superiority of our algorithm in average waiting time, and shows up to 50% improvement to reduce the rejection rate.

6. CONCLUSION

In this paper we have discussed placement and scheduling techniques for heterogeneous reconfigurable computers. An online placement method aimed at reducing the routing cost is presented. Also a parallel scheduling algorithm in hardware and software for minimizing rejection rate is proposed. Moreover, we have investigated the possibility of task preemption on the reconfigurable device and proposed two approaches for saving and restoring the context of preempted hardware tasks.
The placement and scheduling methods were evaluated on a set of randomly generated benchmark and the results have been satisfying. We are currently working on the integration of the methods in an operating system for reconfigurable hardware.

7. ACKNOWLEDGMENTS

This work was supported in part by the German Science Foundation (DFG), SPP 1148 (Rekonfigurierbare Rechensysteme).

8. REFERENCES

[1] A. Ahmadinia, C. Bobda, and T. Jürgen. A New Approach for On-line Placement on Reconfigurable Devices. In *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS-2004), Reconfigurable Architectures Workshop (RAW-2004), IEEE-CS Press,Santa F NM, USA*, April 26-27, 2004.

[2] A. Ahmadinia and J. Teich. Speeding up Online Placement for XILINX FPGAs by Reducing Configuration Overhead. In *Proceedings of the IFIP International Conference on VLSI-SOC, Darmstadt, Germany*, pages 118–122, December, 2003.

[3] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *In IEEE Design and Test - Special Issue on Reconfigurable Computing*, January-March:68–83, 2000.

[4] G. J. Brebner and A. Donlin. Runtime reconfigurable routing. In *IPPS/SPDP Workshops*, pages 25–30, 1998.

[5] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley, 2001.

[6] A. DeHon. Reconfigurable architectures for general-purpose computing. Technical Report AITR-1586, 1996.

[7] H. Simmler, L. Levinson, and R. Manner. Multitasking on FPGA coprocessors. In *FPL*, pages 121–130, 2000.

[8] C. Steiger, H. Walder, and M. Platzner. Heuristics for Online Scheduling Real-time Tasks to Partially Reconfigurable Devices. In *Proceedings of the 13rd International Conference on Field Programmable Logic and Application (FPL'03)*, pages 575–584. Springer, September 2003.

[9] H. Walder, C. Steiger, and M. Platzner. Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS) / Reconfigurable Architectures Workshop (RAW)*, page 178. IEEE Computer Society, April 2003.

[10] G. Wigley and D. Kearney. The development of an operating system for reconfigurable computing. In *Proceedings of the 9th IEEE Symposium Field-Programmable Custom Computing Machines(FCCM'01)*. IEEE-CS Press, April 2001.

Chip Size	56x84	56x84	56x84	80x120	80x120	80x120
Task Size	[20,40]	[20,30]	[25,30]	[20,40]	[20,30]	[25,30]
NPP	564341	919425	1004845	997288	829064	1299699
FF	12428317	1007225	1806210	2139031	2394149	1478445
BF	18033491	3043581	13635866	10125833	3181196	3236485

Table 1: The Routing Cost for different fitters

Scheduling	HW	HW	HW	HW-SW	HW-SW	HW-SW
Task Size	[20,40]	[20,30]	[25,30]	[20,40]	[20,30]	[25,30]
Average Waiting Time(ms)	52	47	46	39	34	35
Rejection Rate	14%	9%	11%	7%	4%	5%

Table 2: Average waiting times and rejection rates