Integrated Temporal Planning, Module Selection and Placement of Tasks for Dynamic Networks-on-Chip

Philipp Mahr, Steffen Christgau, Christian Haubelt
Department of Computer Science
University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam
Email: {*pmahr, hyperion, haubelt*}@cs.uni-potsdam.de

Christophe Bobda
College of Engineering
University of Arkansas
Fayettville, Arkansas 72701
Email: cbobda@uark.edu

***Abstract*—In this work dynamic module selection is integrated in a scheduling and placement flow of tasks for a Dynamic Network-on-Chip. Several implementations (modules) of a task are considered, which differ in size and execution time. In contrast, most state-of-the-art flows consider one module per task, therefore having a static module selection during compile time. Tasks arrive and need to be scheduled and placed by finding a feasible start time and place, such that they meet their deadlines and area requirements. Tasks that do not meet these requirements are rejected. Heuristics for module selection are presented and integrated in an $O(n \log n)$ scheduling and placement flow using EDF-Next-Fit. Evaluation of the dynamic module selection heuristics is performed using synthetic benchmarks. The results show a lower rejection rate of tasks when compared to static module selection.**

I. Introduction

Reconfigurable hardware devices (RHD) allow for dynamic adaption of the computing platform to the application. While early RHDs were limited in size and their reconfiguration capabilities, current state-of-the-art devices offer more than one billion gates and allow partial reconfiguration during runtime. As a consequence, multiple tasks can be executed in parallel and independently on the device. Operating systems for RHDs therefore need to manage the scheduling and placement of tasks during runtime [1], [2]. Tasks, which can range from small filters to complete video decoders, arrive and need to be scheduled by finding a feasible start time, such that they meet their deadlines. Each task is implemented by a pre synthesized rectangular module, which is the actual hardware being placed on the RHD. A start time can only be assigned to a task, if the module can be placed on the device. Therefore, a strong nexus between the temporal planning and placement of tasks exists. Most works in the area of scheduling and placement for RHDs consider only static module selection, meaning that the module to be placed on the RHD is known at compile time [1], [3], [4], [5], [6].

However, tasks can be computed by different modules varying in size, execution time and communication overhead: When for instance computing a fast fourier transformation (FFT) the basic operation is the butterfly operation. A N-point DFT needs $log_2(N)$ stages with $N/2$ radix-2-butterflies per stage. Feasible FFT implementations need to calculate one radix-2-butterfly operation at minimum, covering the area a, or at most $N/2$ butterflies covering an area of roughly $a \cdot N/2$, not considering the area needed for communication. Thus the execution time of a N-point DFT varies when considering different modules. The largest module computes $N/2$ butterflies in parallel and is about $N/2$ times faster then the smallest module. Other applications from image processing, sorting algorithms or applications using the Monte Carlo method show a similar behavior. The availability of several modules with different characteristic implementing the very same task, does not only expand the design space for static designs but also for dynamic implementation like Dynamic Networks-on-Chip. In particular, selecting the best module has impact on both schedulability (including placement) and optimality of an implementation. Hereby, the quality is typically measured in terms of rejection rate.

In this paper, we propose an approach that integrates temporal planning, module selection, and placement of tasks. Different heuristics for on-line module selection and a comprehensive evaluation is presented by comparing our proposed approach to non-integrated approaches, which perform on-line task scheduling and placement but off-line module selection.

A. Related Work

Most publications on managing RHDs are focused on task scheduling and placement, performing module selection off-line or assuming only a single module per task. Work on scheduling and placement can be divided into off-line and on-line methods targeting both homogeneous or heterogeneous reconfigurable resources. Danne and Platzner [5] consider off-line scheduling of periodic real-time tasks and adapted the global EDF scheduling approach to homogeneous reconfigurable hardware. Two preemptive scheduling algorithms, EDF-First-k-Fit and EDF-Next-Fit, are presented. The influence of reconfiguration overhead is evaluated and results show, that reconfiguration times one or more orders of magnitude smaller than task computation time does introduce only small losses in performance. However, their work is based on a simple resource model and does not consider any placement algorithms.

Bazargan et al. [3] were one of the first authors to introduce on-line placement algorithms for two-dimensional homogeneous reconfigurable hardware. Their algorithm partitions the

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.150

free-space and maintains a set of all maximal free rectangles, which leads to a quadratic size of the set of free rectangles. Alternatively, a method partitioning the free-space into only $O(n)$ rectangles using heuristics is proposed, which in comparison shows a higher rejection rate when considering the same task set. Steiger et al. [1] improved the Bazargan's $O(n)$ partitioner by delaying the split decision of the resulting free rectangles after a module has been placed. [4] showed, that the free space in the form of maximal empty rectangles can be managed more efficiently by using the staircase data structure. In [6] an algorithm considering routing conscious placement is presented, which manages the free space in $O(n \log n)$, by using plane sweep methods from computational geometry. Also, the decision were in the free space to optimally place a module is calculated using weighted communication costs instead of considering the size of the free rectangles. In [7] an on-line placement method for heterogeneous devices using a discrete hopfield neuronal network is presented. The neural network outperforms the older SUF Fit algorithm presented in [8]. When placing modules on a reconfigurable device the free space becomes fragmented as tasks finish, which can lead to a higher rejection rate of tasks, even though the sum of the free space is higher then the area requirements of the task. [9] presents a defragmentation approach for one dimensional heterogeneous reconfigurable devices in a no-break fashion, by copying one module at a time and then switch the running computation to the copied module. Results showed that the presented approach lead to larger free space of up to 50%. In [10] Tabero et al. present a metric able to estimate the fragmentation status of a reconfigurable device by analyzing the free space. Heuristics to decide when and how to perform a defragmentation of the device are presented. [11] uses the metric to implement a fragmentation-based heuristic to decide were a task in the free space should be placed. In [12] an off-line placement algorithm of tasks considering geometrical task variants is presented. Tasks are modeled as three dimensional boxes given by their width, height and execution time. Modified heuristic methods from floorplanning to select the task variants are applied, leading to better solutions for the placement problem. In contrast to the above described publications, our approach integrates temporal planning, dynamic module selection and placement. This approach allows to react dynamically to arriving tasks as well as the current state of the platform. Besides deciding what task should be planed next and deciding when to place a task, dynamically selecting a module improve the overall implementation in terms of rejection rate and finishing time. The presented approach is shown in Figure 1.

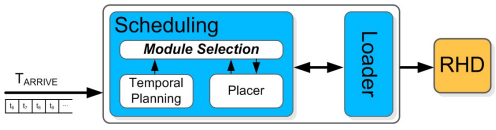

Fig. 1. Resource management with integrated module selection

The remainder of this paper is organized as follows: Section II presents the resource and task model and gives a formal problem definition. The proposed method of integrated temporal planning, module selection ans placement is presented in Section III. The simulation environment and evaluation of the proposed approach is given in Section IV providing a comparison to off-line module selection methods. Finally, Section V concludes this paper.

II. Resource, Tasks & Scheduling

In this section a formal problem definition is given. This will be used as a foundation to present the proposed approach of integrated temporal planning, module selection and placement. A reconfigurable device generally consists of a number of reconfigurable units (RCUs). RCUs define the granularity of the device ranging from fine-grained RCUs, consisting mainly of registers, multiplexers and look-up tables (FPGA logic blocks), to coarse-grained RCUs which consist of complete ALUs able to perform computation on word-level. RCUs are in the majority of cases arranged in a rectangular grid of the area $A_x \cdot A_y$. Between RCUs an interconnection network exist. Both the interconnect and the RCU are reconfigurable during runtime. Tasks need to be executed by the reconfigurable hardware device. Each task t_i holds an associated deadline d_i and can have communication points somewhere on the device, which models the communication with memories or peripherals at the border of the device. Inter task communication is considered by supporting communication points between tasks. Task preemption is allowed and two preemption models are considered. With the first model a preempted task needs be restarted from the beginning, while the second model allows the task to continue after it has been placed again. Each task has a set of rectangular modules M_i, which consists at least one module. The modules are pre synthesized and represent the hardware implementation of the task covering one or more RCUs. All modules are able to execute the task but differentiate in size and execution time. The DyNoC computing architecture is considered for this work.

A. Dynamic Network-on-Chip

In [13], [14] Bobda et. al. presented the DyNoC, a reconfigurable computing architecture for the dynamic interconnection of reconfigurable modules in a mesh network. In the basic state with no modules placed, the network behaves like a normal network-on-chip (NoC), consisting of processing elements and routers. Processing elements (PEs) access the network via corresponding routers. In contrast to a normal NoC, PEs can also communicate with their nearest neighbors using direct links. This allows the aggregation of several PEs to create a rectangular module, to compute a complex task. Such modules can be dynamically placed and removed during run time. PEs inside a module do not need to use the routers in between for communication, but use the direct links instead. The advantage of this hybrid communication scheme is a gain in flexibility. Figure 2 shows the DyNoC system architecture, whereas

modules M1, M2 and M3 covering several PEs and routers are placed on the network.

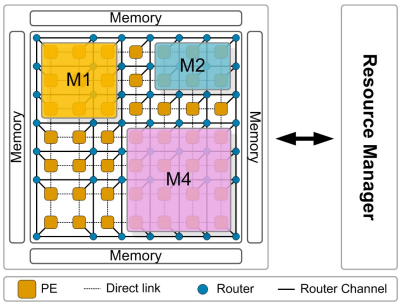

Fig. 2. DyNoC system architecture

In the DyNoC a RCU consists of one processing element and an associated router. With tasks having a set of rectangular modules covering several RCUs the relation between tasks and modules is defined as follows:

Definition (Task-Module-Relation): *For all tasks $t_i \in T$ a non empty set M_i of modules $m_j \in M_i$ exist.*

With each module $m_j \in M_i$ an area requirement w_j (width), h_j (height) and a worst-case execution time c_i^j for the task t_i with the deadline d_i is associated. The task has to be mapped on the reconfigurable device depending on his priority, as described in Section III-B. Thus, when considering several modules able to execute a task the selection of a module should take place first.

Definition (Feasible Module Selection): *A feasible module selection assigns a module $m_j \in M_i$ to t_i, so that a feasible schedule exist.*

A feasible schedule has to consider both spatial and temporal aspects, which is why a place on the device as well as a start time (temporal planning) for a task have to be found. The problem of placement and temporal planning are strongly related, so that a feasible start time can not be found without considering the placement of the module.

Definition (Feasible Placement): *Given a set M_p of already placed modules $m_k \in M_p$. Associated with all modules $m_k \in M_p$ is an origin x_k, y_k and the required space w_k, h_k of the modules. A feasible spatial planning for a new arrived task t_i at time a_i with the selected module m_j and the required space w_j, h_j exist, if a position x_j, y_j on the reconfigurable device exists, which fulfill the following conditions:*

I) $x_j + w_j \leq A_x \wedge y_j + h_j \leq A_y$

II) $\forall m_k \in M_p$*:*

$$x_k + w_k \leq x_j \wedge y_k + h_k \leq y_j \wedge$$
$$x_j + w_j \leq x_k \wedge y_j + h_j \leq y_k$$

When a feasible placement for module m_j exist a start time $s_i \leq a_i$ can be assigned to the task t_i. Note that modules which finished their execution can be removed from the RHD, i. e., $M_p = M_P \setminus m_k$.

Definition (Feasible Temporal Planing): *A feasible temporal planning assigns a task t_i a start time s_i, so that $s_i + c_i^j < d_i$ for the selected module $m_j \in M_i$ holds.*

Definition (Feasible Schedule) *A feasible schedule exist, when a placement and a temporal planning for the task exist.*

Problem Definition: *Given a set T of tasks $t_i \in T$ with arrival times a_i and associated modules $m_j \in M_j$.*[1] *For each task $t_i \in T$ select a module m_j such that a feasible schedule for the maximal number of tasks t_i exists.*

In section IV this is evaluated by calculating the rejection rate of the task sets.

III. Integrated temporal planning, module selection and placement of tasks

In this section the integrated temporal planning, module selection and placement is presented. We start by describing placement and temporal planning.

A. Placement

Placement generally consists of two steps. The partitioner or free space manager and the fitter. The partitioner keeps track of all free rectangles on the device while the fitter selects one free rectangle depending on a selection strategy. In the work at hand the routing-conscious dynamic placement algorithm from Ahmadinia et al. [6] is used. This algorithm can be computed with $\theta(n \log n)$ and considers the manhatten distance between modules to select the optimal placement. Therefore the weighted communication costs are minimized. Compared to [3], [15], [11] the algorithms in [6] manage the free space by storing the occupied space and finding a placement for a single point instead of searching through all free rectangles. This is done by shrinking the reconfigurable hardware and simultaneously blow up the placed modules M_p by half of the width $x_j/2$ and half of the height $y_j/2$ of the module to be placed m_j. Thus, all possible positions for m_j are reduced to points instead of rectangles. The points on the contour are of particular interest as they preserve a good structure of the free space. The contours of the free space are computed using the CUR algorithm (contour of union of rectangles) [16] from computational geometry, which performs plane sweeps and has a segmentation tree as a data structure. The best spot for the module in the free space is found by minimizing the manhatten distance between communication points.

[1] In this work, only independent tasks are considered in order to evaluate the achievable performance through integrated temporal planning, module selection and placement. In future work, this approach will be extended to support communicating tasks.

B. Temporal Planing

When a feasible placement for a task exist and the task t_i will meet its deadline d_i (feasible temporal planning) the task is accepted and will be placed on the reconfigurable hardware.

When considering dynamic arrival of hard real-time tasks with their area requirements, execution times and deadlines one has to order these tasks in a priority queue according their requirements. Then the tasks are scheduled (find a feasible placement and make a temporal planning) one by one from the queue. Steiger et al. [1] have shown that sorting the tasks according to their deadline lead to better results compared to sorting according to their size. This sorting process is the adaption of the well known Earliest Deadline First (EDF) dynamic scheduling algorithm for real-time systems. Furthermore they considered putting an currently unplaceble task on hold and continue with the next task in the queue until the device allocation changes. Danne [5] called this the EDF-NF (Next-Fit) algorithm and gave a schedulability test for periodic task sets. With the Next-Fit strategy the tasks can be placed and executed out-of-order, thus allowing the reconfigurable hardware to be better utilized. In the work at hand we use the EDF-NF, which assigns a priority based on the deadline.

C. Module Selection

State-of-the-art on-line scheduler only considered one module per task. But with several modules able to execute a task a module has to be selected during runtime. Selecting the right module is crucial when considering device utilization and rejection rate. A good module selection strategy therefore should be dynamic in order to allow adaption to the current utilization of the device or timing requirements of the task. Another advantage of having multiple modules per task is the possibility of considering other modules of a task, when the chosen one cannot be planed.

1) Dynamic Module Selection: The *Largest Module First* (LMF) heuristic, which considers the module with the largest area requirements first. When the largest module can not be placed, because the free space is not large enough, the second largest module is considered. The placement is invoked again until a feasible placement for the task has been found or the execution time of the chosen module excesses the deadline. This strategy tries to compute a task as fast as possible.

Smallest Module First (SMF) is the counter part to LMF and considers the module with the smallest area requirements first. When the runtime of this module excesses the deadline the next largest module is chosen until the smallest module has been found, which then meets the deadline of the task. This strategy tries to maximize the free space by using only the smallest module.

2) Static Module Selection: While dynamic selection heuristics allow the adaption to the device state or timing by selecting smaller (slower) or larger (faster) modules, static heuristics preselect a module for placement at compile time and do not take other modules of a task into account. The following static module selection heuristics are considered as a comparison to the dynamic selection heuristics.

Largest Module Only (LMO) considers the module with the largest area requirements only.

The *Smallest Module Only* (SMO) heuristic chooses the module with the smallest area requirements which meets the deadline d_i.

The *Average Size Module Only* (ASMO) takes the module with the average size.

D. Extended Scheduler

When a task arrives it is put in the $T_{FLOATING}$ list and the scheduler is called. $T_{FLOATING}$ holds all tasks, which have to be placed on the device and is sorted according to a scheduling algorithm, which is EDF-Next-Fit in this case. The first task t_i of this list is taken and checked for feasibility. A task is feasible, if there is at least one module, which meets the deadline $t_i.deadline$. When the task is not feasible, it is rejected and put into the $T_{REJECTED}$ list. A module m_j of the feasible task is selected according to the module selection strategy and the placer is invoked. When placement is successful the task is loaded on the device and put into the $T_{RUNNING}$ list, otherwise the next module of the task is considered depending on the module selections strategy. When no feasible module can be placed, the next task is considered (next fit) and t_i is put back in $T_{FLOATING}$, because there is still the possibility, that the task can be scheduled later. Algorithm 1 shows the pseudo code of one run of the scheduler.

Algorithm 1 Scheduling

```
success ← FALSE
k ← 0
Sort(T_FLOATING)
while !success && k <= next_fit do
  t_i ← Pop(T_FLOATING)
  M_s ← t_i.M_j
  if M_s.MinRuntime() + time < t_i.deadline then
    while !success && M_s ≠ ∅ do
      m_j ← SelectModule(M_s)
      success ← PlaceModule(m_j)
    end while
    if success then
      t_i.start_time ← time
      t_i.state ← RUNNING
      T_RUNNING ← t_i ∪ T_RUNNING
    else
      t_i.state ← FLOATING
      T_FLOATING ← t_i ∪ T_FLOATING
    end if
  else
    t_i.state ← REJECTED
    T_REJECTED ← t_i ∪ T_REJECTED
  end if
  k ← k + 1
end while
```

The algorithm has an overall complexity of $O(n \log n)$. In [6] Ahmadinia et al. showed that the placement has a complexity of $O(n \log n)$. All other steps of the algorithm are simply sorting lists which also takes $O(n \log n)$ or accessing

lists which is linear. Thus the overall complexity of the scheduling algorithm is still in the $O(n \log n)$ domain.

IV. Evaluation

An event-based simulator has been implemented to evaluate the scheduling flow. An overview about the simulator is given in Figure 3.

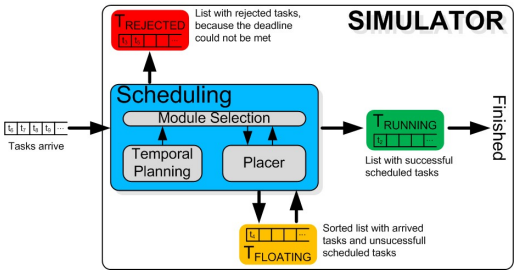

Fig. 3. Simulator

The performance of the module selection strategies is evaluated using the percentage of rejected tasks, the wait time, which is the time between the arrival and actual start time of the task and the average earliness. The earliness describes the a time task finishes prior its deadline. All task sets are evaluated using EDF-Next-Fit scheduling with next-fit values of $k \in \{0, 1, 3, 10, \infty\}$ and a device size of 100×100. No communication and precedence between tasks was considered, as only the influence of selecting modules dynamically is evaluated. Several task sets with different states of device utilization were benchmarked and three representative task sets with high device utilization were chosen. Each task set TS1, TS2 and TS3 contains 500 tasks and all modules of a task were created by generating one rectangular module with dimensions between 4 and 32 RCUs for x and y and a random runtime between 16 and 48 time steps. From this module all other modules for the task were derived by increasing and decreasing the size and calculating the runtime depending on the change of size. The deadline of the task was randomly chosen, with the constraint that the runtime of all modules were able to meet the tasks deadline. Arrival times of tasks were randomly chosen between 0 and 500 time steps. The main difference between the task sets TS1, TS2 and TS3 is the amount of modules per task. A task in TS1 has seven modules, while a task in TS3 only has three modules.

Table I shows the rejection rate of the three task sets. TS1 has seven modules per task ranging from 0.04% to 54.7% area requirements. Dynamic module selection using LMF leads to no task rejections in TS1, while SMO, the runner-up, lead to a rejection rate of 9.0% in comparison. When allowing out-of-order placement by increasing k, the difference between all strategies shrinks, but still LMF has the advantage. For TS2 and TS3 less modules per task were available limiting the selection flexibility and therefore leading to an overall worse rejection rate for all module selection strategies. While the LMF strategy has no tasks rejected in TS1, with TS2 and TS3 the reconfigurable hardware becomes highly utilized

TABLE I
Rejection rate for three task sets

task set	k	LMF	LMO	SMF	SMO	ASMO
	0	0.0%	33.6%	62.2%	9.0%	9.2%
TS 1	1	0.2%	16.0%	62.4%	6.8%	6.6%
0.04% -	3	0.2%	8.0%	54.8%	6.4%	6.4%
54.7%	5	0.4%	7.8%	3.8%	6.4%	6.4%
7 mod.	10	0.2%	5.2%	4.4%	6.4%	6.4%
	∞	0.2%	3.8%	4.0%	6.4%	6.4%
	0	16.0%	35.4%	58.0%	15.8%	16.0%
TS 2	1	6.6%	16.4%	56.2%	11.0%	11.2%
0.06% -	3	3.4%	10.8%	45.8%	10.2%	10.0%
36.0%	5	3.0%	7.6%	34.8%	10.0%	10.0%
5 mod.	10	3.4%	6.8%	7.8%	10.0%	10.0%
	∞	5.6%	7.0%	8.8%	10.0%	10.0%
	0	23.2%	24.0%	85.0%	16.6%	14.8%
TS 3	1	14.6%	16.6%	78.6%	14.8%	14.8%
0.09% -	3	9.2%	11.8%	67.0%	13.6%	13.6%
18.5%	5	8.8%	10.0%	36.4%	13.6%	13.6%
3 mod.	10	8.0%	9.8%	11.8%	13.6%	13.6%
	∞	8.6%	9.0%	11.0%	13.6%	13.6%

(saturated) and the situation changes. LMF basically behaves as follows. First larger modules are placed and the free space between the large modules is filled-up with smaller modules. When a large module finishes a module with at most the size of the finished module can be placed. Generally the module placed will be smaller creating a small area of free space which can be filled by small modules, if available. With SMO the free space between placed modules is smaller allowing no modules being placed in between, thus showing a better utilization of the device, as seen in TS2 and TS3. However, this behavior occurs mainly when the reconfigurable hardware is highly utilized. When the device utilization is lower, less tasks need to be placed and therefore with LMF the free area consists of larger parts. Larger modules can be placed and the average earliness of tasks is maximized. The rejection rate is lower because LMF in this case is more responsive compared to SMO, which can be seen in TS1. SMF shows poor performance compared to SMO and LMF for all task sets. One would expect SMF to show similar performance than SMO. However, when the smallest module of a task can not be placed and does not meet the deadline anymore the task is rejected with SMO. With SMF only the module is discarded and the next larger module (which still meets the deadline) is considered. Due to the fact, that the smallest module could not be placed it is unlikely that a larger module can be placed. So the task occupies a high priority place at the floating task list, which is why SMF shows poor performance for small k.

The earliness of a task is the time a task finishes in relation to its deadline and is positive, if the task finishes prior its deadline and negative otherwise. In Table II the average earliness of the accepted tasks is considered. For all task sets selecting larger modules with LMF and LMO lead to tasks finishing earlier.

In Table III the average wait time of tasks is shown. While with SMO almost all tasks are started instantly for $k = 0$, larger k lead to earlier start times for all selection strategies.

Dynamic modules selection as well as out-of-order place-

TABLE II
AVERAGE EARLINESS OF ACCEPTED TASKS FOR THREE TASK SETS

task set	k	LMF	LMO	SMF	SMO	ASMO
	0	80.6	50.7	8.5	7.5	7.5
TS 1	1	80.8	48.8	8.4	7.7	7.7
0.04% -	3	76.7	52.8	8.4	7.9	7.9
54.7%	5	74.7	58.0	14.9	7.9	7.9
7 mod.	10	74.0	71.4	16.0	7.9	7.9
	∞	74.0	116.7	8.6	7.9	7.9
	0	61.9	37.3	7.8	6.0	6.0
TS 2	1	56.3	42.8	8.0	6.2	6.2
0.06% -	3	56.9	50.3	8.9	6.4	6.4
36.0%	5	58.8	54.2	10.6	6.4	6.4
5 mod.	10	62.1	66.6	10.0	6.4	6.4
	∞	64.3	97.4	6.6	6.4	6.4
	0	11.5	10.8	2.7	2.8	2.8
TS 0	1	12.5	13.6	3.0	2.8	2.8
0.09% -	3	15.6	16.3	3.5	2.8	2.8
18.5%	5	19.8	20.1	5.7	2.8	2.8
3 mod.	10	20.8	29.1	4.0	2.8	2.8
	∞	21.4	33.0	3.5	2.8	2.8

TABLE III
AVERAGE WAIT TIME OF ACCEPTED TASKS FOR THREE TASK SETS

task set	k	LMF	LMO	SMF	SMO	ASMO
	0	14.2	94.6	4.5	0.6	0.5
TS 1	1	11.9	103.6	3.0	0.3	0.4
0.04% -	3	5.9	103.6	12.5	0.2	0.1
54.7%	5	1.7	98.1	52.6	0.1	0.1
7 mod.	10	1.8	87.1	14.4	0.1	0.1
	∞	1.8	41.1	2.5	0.1	0.1
	0	24.9	75.4	6.8	0.6	0.5
TS 2	1	33.9	75.0	7.9	0.3	0.4
0.06% -	3	29.2	69.9	15.3	0.2	0.2
36.0%	5	22.2	66.9	15.3	0.3	0.3
5 mod.	10	10.3	55.5	24.4	0.3	0.3
	∞	1.1	24.2	1.3	0.3	0.3
	0	25.1	28.8	1.5	0.2	0.2
TS 3	1	24.5	27.6	4.6	0.1	0.1
0.09% -	3	17.9	26.1	6.8	0.2	0.2
18.5%	5	8.0	22.5	8.9	0.2	0.1
3 mod.	10	2.6	14.1	2.4	0.2	0.1
	∞	1.6	10.1	1.6	0.2	0.1

ment leads to an increased amount of the total number of placements per task set. For TS1 LMF needs 4132 placements for $k = 0$ and 12552 placements for $k = \infty$, while SMO needs only 669 placements for $k = 0$ and 931 placements for $k = \infty$. However, in this case SMO rejects 9% of the tasks. For TS3 the placer is invoked between 1020 and 10961 times for LMF and between 609 and 957 times for SMO. For a new task set TS1*, which is similar to TS1 but has a lower runtime (between 16 and 24) and thus a lower device utilization allowing all module selections strategies to process all tasks successful (0% rejection rate), LMF needs 1143 placements for all k while SMO needs 500 placements for all k.

V. CONCLUSION

In this paper we propose to integrate dynamic module selection into an on-line flow for scheduling and placement of tasks on reconfigurable hardware. Having several modules implementing the same task, differing in size and execution time, allows to adapt to the task execution and device state. Heuristics for the dynamic module selection have been introduced and evaluated showing good results in terms of rejection rate, average earliness and wait time. Future work will focus on the extension of the model to support task precedence, configuration time and communication costs. Module resizing and replacement during runtime is expected to lead to a further reduction of the rejection rate, minimization of communication distances and lower fragmentation.

REFERENCES

[1] C. Steiger, H. Walder, M. Platzner, and L. Thiele, "Online scheduling and placement of real-time tasks to partially reconfigurable devices," in *In: Proceedings of the 24th International Real-Time Systems Symposium, Cancun*, 2003, pp. 224–235.

[2] G. B. Wigley and D. A. Kearney, "Research issues in operating systems for reconfigurable computing," in *In Proceedings of the International Conference on Engineering of Reconfigurable System and Algorithms(ERSA*. CSREA Press, 2002, pp. 10–16.

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

[4] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online fpga placement," in *DAC'04: Proceedings of the 41st Design Automation Conference*, 2004.

[5] K. Danne and M. Platzner, "An edf schedulability test for periodic tasks on reconfigurable hardware devices," in *LCTES '06: Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*. New York, NY, USA: ACM, 2006, pp. 93–102.

[6] A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. C. van der Veen, "Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices," *IEEE Trans. Comput.*, vol. 56, no. 5, pp. 673–680, 2007.

[7] A. Eiche, D. Chillet, S. Pillement, and O. Sentieys, "Task placement for dynamic and partial reconfigurable architecture," in *DASIP '2010: Proceedings of the 2010 Conference on Design & Architectures for Signal & Image Processing*. Piscataway, NJ, USA: IEEE Press, 2010, pp. 642–649.

[8] M. Koester, M. Porrmann, and H. Kalte, "Task placement for heterogeneous reconfigurable architectures," in *FPT*, 2005, pp. 43–50.

[9] S. Fekete, T. Kamphans, N. Schweer, C. Tessars, J. van der Veen, J. Angermeier, D. Koch, and J. Teich, "No-break dynamic defragmentation of reconfigurable devices," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, Heidelberg, Germany, Sep. 2008, pp. 113–118.

[10] J. Septin, H. Mecha, D. Mozos, and J. Tabero, "2d defragmentation heuristics for hardware multitasking on reconfigurable devices," in *IEEE Reconfigurable Workshop (RAW), Proceedings of the International Parallel and Distributed Processing Symposium*, 2006.

[11] J. Tabero, J. Septin, H. Mecha, and D. Mozos, "A low fragmentation heuristic for task placement in 2d rtr hw management." in *FPL*, ser. Lecture Notes in Computer Science, J. Becker, M. Platzner, and S. Vernalde, Eds., vol. 3203. Springer, 2004, pp. 241–250. [Online]. Available: http://dblp.uni-trier.de/db/conf/fpl/fpl2004.htmlTaberoSMM04

[12] K. Danne and S. Stuehmeier, "Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants," 2005.

[13] C. Bobda and A. Ahmadinia, "Dynamic interconnection of reconfigurable modules on reconfigurable devices," *IEEE Design and Test of Computers*, vol. 22, pp. 443–451, 2005.

[14] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. P. Fekete, and J. van der Veen, "Dynoc: A dynamic infrastructure for communication in dynamically reconfigurable devices," in *FPL*, 2005, pp. 153–158.

[15] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1393–1407, 2004.

[16] R. H. Güting, "An optimal contour algorithm for iso-oriented rectangles," *J. Algorithms*, vol. 5, pp. 303–326, September 1984. [Online]. Available: http://portal.acm.org/citation.cfm?id=1807.1974