Scheduling Temporal Partitions in a Multiprocessing Paradigm for Reconfigurable Architectures

Popp, Andreas; Le Moullec, Yannick; Koch, Peter

Published in:
NASA/ESA Conference on Adaptive Hardware and Systems, 2009. AHS 2009

DOI (link to publication from Publisher):
10.1109/AHS.2009.43

Publication date:
2009

Document Version
Accepted author manuscript, peer reviewed version

Scheduling Temporal Partitions in a Multiprocessing Paradigm for Reconfigurable Architectures

Andreas Popp, Yannick Le Moullec, and Peter Koch
Center for Software Defined Radio & Technology Platforms Section,
Department of Electronic Systems, Aalborg University
Aalborg, Denmark
{anp,ylm,pk}@es.aau.dk

***Abstract*—In this paper we describe a mapping methodology for heterogeneous reconfigurable architectures consisting of one or more SW processors and one or more reconfigurable units, FPGAs. The mapping methodology consists of a separated track for a) the generation of the configurations for the FPGA by level-based and clustering-based temporal partitioning, and b) the scheduling of those configurations as well as the software tasks, based on two multiprocessor scheduling algorithms: a simple list-based scheduler and the more complex extended dynamic level scheduling algorithm. The mapping methodology is benchmarked by means of randomly created task graphs on an architecture of one SW processor and one FPGA. The results are compared to a 0-1 integer linear programming solution in terms of exploration time as well as the finish-time of all tasks of the application. The results show that, in 90% of the investigated cases, the combination of level-based temporal partitioning and extended dynamic level scheduling gives the best performance in terms of finish-time of the full task-set.**

***Keywords*-Reconfigurable Hardware; Heterogeneous Reconfigurable Architectures; Temporal Partitioning; Multiprocessor Scheduling**

I. Introduction

Most signal processing architectures are both reconfigurable and heterogeneous, consisting of several software processors as well as configurable hardware, typically Field-Programmable Gate Arrays (FPGAs). Moreover, FPGAs provide reconfiguration during runtime, either for the full FPGA area - or for a portion of the area, noted Dynamic Partial Reconfiguration (DPR). Such systems have the possibility to provide better performance than compile-time configured systems in terms of total execution time, logic resource usage, and power consumption [1]. However, in order to obtain such performance benefits, it is necessary to have efficient scheduling techniques and methods which we denote "mapping methods" in the following.

Existing solutions for mapping applications to reconfigurable heterogeneous architectures target architectures consisting of a software processor connected to a reconfigurable FPGA via a common bus. The software processor serves as the host, either being 1) a simple configuration controller for the reconfigurable hardware, or 2) a processor that utilizes the reconfigurable hardware for acceleration of computationally heavy tasks.

In case 1 where the processor works solely as a configuration controller, approaches for temporal partitioning have been suggested by, among others, Kaul&Vemuri [2] and Purna&Bhatia [3]. Temporal partitioning is the task of dividing a large application into partitions that are mutually exclusive in time, and thus can be executed sequentially on a device that is smaller than needed for fully parallel implementation of the entire application.

In case 2, approaches have been suggested by, among others, Banerjee et al. [4] that formulated the solution as a 0-1 Integer Linear Programming (ILP) problem to obtain the minimum cost in terms of overall execution time. Noguera&Badia [5] proposed a HW/SW partitioning algorithm where tasks are moved between HW and SW until a minimum overall execution time is obtained. The method considers prefetching of configurations to reduce the reconfiguration overhead. The computational complexity of both works is high (non-polynomial for the first), leading to prohibitively long execution times of exploration algorithms, which we from hereon will denote "exploration times". An approach with lower computational complexity has been proposed by Chatha&Vemuri [6]. The work consists of an algorithm of five steps: a) HW/SW partitioning, b) temporal partitioning of HW tasks, c) scheduling of HW and SW tasks, d) scheduling of HW reconfigurations, and e) scheduling of communications.

However, as these three approaches do cover a subset of heterogeneous reconfigurable architectures, they are not suited for architectures consisting of several units, both in HW and SW.

Mapping methods for homogeneous SW architectures have been well studied for some time. One of the well known methods is Dynamic Level Scheduling (DLS) by Sih&Lee [7], who in the same connection propose an extended DLS algorithm for heterogeneous architectures.

The previously mentioned approaches do not cover heterogeneous reconfigurable architectures consisting of several processing units, thus in this work we combine the known temporal partitioning algorithms with multiprocessor scheduling algorithms in a scheduling methodology for heterogeneous reconfigurable architectures. The methodology is inspired by Chatha&Vemuri [6] that starts with an initial HW/SW partitioning followed by creation of temporal partitions for HW nodes. The temporal partitions are then treated as super-nodes in a multiprocessing framework - where the super-nodes are tied to a particular unit, the reconfigurable HW unit.

This is the author's version of the paper published in
NASA/ESA Conference on Adaptive Hardware and Systems, 2009

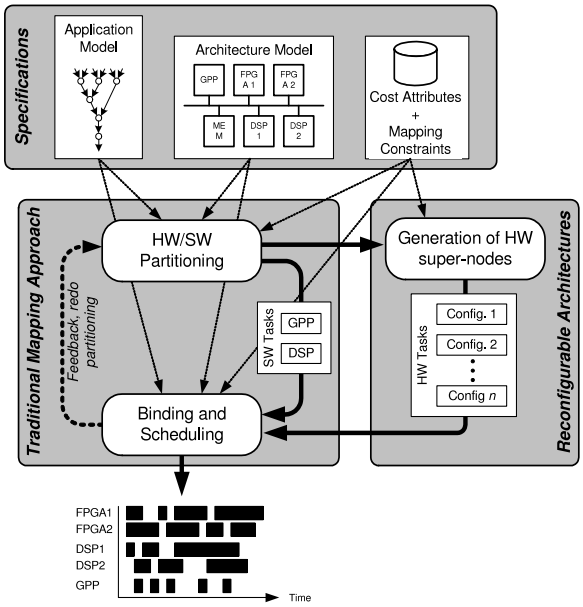

Fig. 1. The proposed mapping methodology. The first step is the specification of the application, architecture, and their interrelation via a cost-library. This is followed by a partitioning between HW and SW tasks. The HW tasks are sent to the HW-flow, where the tasks are partitioned into temporal partitions of HW tasks. The HW tasks and their reconfiguration are each considered super-nodes of tasks, which are fed to the multiprocessor binding and scheduling process.

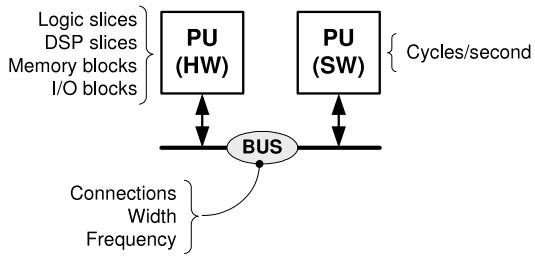

Fig. 2. General Architecture Model. The attributes for each architecture element is found by studying the data sheets of the architecture.

This paper describes the suggested methodology in section II, including the underlying application and architecture model. This is followed by a series of experiments in section III where the mapping results are compared to a 0-1 ILP solution that serves as a lower boundary reference. The results are presented in section IV, followed by a discussion and a conclusion in section V and VI, respectively.

II. Mapping Methodology

The proposed mapping methodology is a combination of multiprocessor scheduling and temporal partitioning for reconfigurable architectures, and is outlined in figure 1. The starting point is the specifications of the application, architecture, and cost-library which are all expanded in section II-A. Following the specification, the application's tasks are partitioned between HW and SW units, and between several HW units. This is fed back to the original SW multiprocessor scheduling flow, as described in section II-D.

A. Specifications and Modeling

The application is specified as a directed acyclic task-graph, consisting of nodes and edges. The nodes represent tasks, whereas the edges represent data dependencies. The edges are assigned a width, describing the amount of data transferred between the nodes. The task granularity can vary, being both single algorithmic operations as well as larger blocks of operations. The general architecture model is illustrated in figure 2. The model is a composition of Processing Units (PUs), memories, and ports, all connected via buses. PUs are again either SW or HW. SW units have a certain number of cycles per second, whereas HW has a number of resources, each corresponding to the number of logic slices, DSP resources, memory blocks etc. Buses are described by the units they connect, their direction, width, and frequency.

The cost-library binds the application and the architecture together. It contains the cost of various implementation alternatives for each task, i.e. execution time for SW and execution time, reconfiguration time, and resource usage for HW. Reconfiguration time is derived from the size of the reconfigurable HW. The cost-library is derived by sample implementations of each task, without having to perform the full implementation of the application. Another option is to provide estimates based on previous experiences.

B. Partitioning

The partitioning approach is based on the values of the cost-library: t_i^{sw} is the SW execution time of task i, t_i^{hw} is the hardware execution time of task i, and $t_{\mathrm{reconf}}^{\mathrm{hw}}$ is the full reconfiguration time of the HW unit. The HW/SW partitioning is based on the principles described in the list below:

1) If logic slice resource usage for task i is larger than the capacity of the HW unit, then partition to **SW**
2) Else If $t_{\mathrm{reconf}}^{\mathrm{hw}} + t_i^{\mathrm{hw}} < t_i^{\mathrm{sw}}$, then partition to **HW**
3) Else $t_{\mathrm{reconf}}^{\mathrm{hw}} + t_i^{\mathrm{hw}} \geq t_i^{\mathrm{sw}}$ is true, so partition to **SW**

As seen in the partitioning scheme, reconfiguration time is included in the HW execution time, assuming that each HW execution must be preceded by reconfiguration.

C. HW Flow

The partitioning is followed by an extraction of the HW tasks from the application graph.

The task-set is then temporally partitioned, following the two list-scheduling temporal partitioning algorithms by Purna&Bhatia [3]. However, it is a requirement to the execution scheme that temporal partitions do not start execution before all inputs are ready. Thus, there must not be a path through other nodes or partitions from an output to an input in the same partition. Therefore, the temporal partitioning algorithms are extended with a search for paths outside the current partition. If such a path exists, a new partition is created, and the current node is placed in that new partition.

The result of the HW flow is fed to the binding and scheduling by performing an application graph and cost-table update. In the application graph update, the temporal partitions are considered as HW super-nodes, and are fed to the SW flow

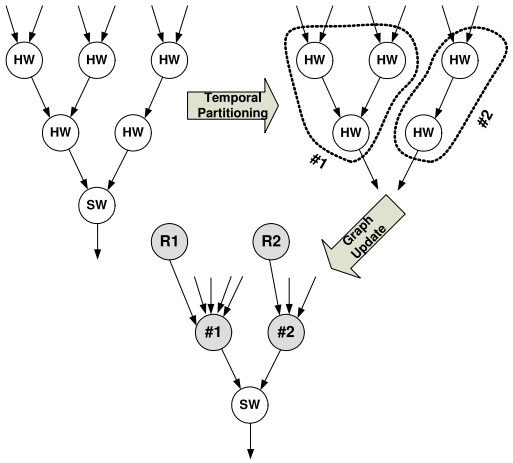

Fig. 3. Illustration of the application graph update. Firstly, HW nodes are temporally partitioned. Secondly, nodes in temporal partitions are replaced by super-nodes, followed by insertion of reconfiguration nodes for each super-node.

as super nodes. The cost-table entries for the HW tasks are removed and replaced by cost-table entries for the super-nodes.

The application graph and cost-table update follows the scheme as described below and refers to the illustration of the application graph update in figure 3:

1) All nodes in the same temporal partition are replaced by a single super-node (**#1** and **#2** in figure 3). This is performed for all temporal partitions. All edges going to/from those nodes are being redirected to the corresponding super-nodes, preserving the direction of the edge.
2) Reconfiguration nodes (**R1** and **R2**) are added to all the new super-nodes. The reconfiguration nodes have no predecessors, and their only successor will be the corresponding super-node.
3) The cost-table is updated by firstly removing the entries for the nodes that are replaced by super-nodes. Secondly, entries are added for each super-node. The execution time is the maximum execution time of the tasks in the super-node. The resource cost is the sum of all tasks in the super-node.
4) Similarly, entries are added for the reconfiguration nodes. The execution is similar to the reconfiguration time of the unit, and the resource cost is similar to the super-node that is reconfigured.

D. SW Flow

The SW scheduling flow is based on two approaches:

1) A simple list-scheduler where nodes are scheduled in the order given by the finish-time of their predecessor as well as their mobility, such that the node with the lowest mobility is scheduled first.
2) The extended DLS algorithm by Sih&Lee [7] for heterogeneous processor systems.

For both approaches additional constraints have been included in order to ensure that reconfiguration and execution sequences are performed in the right order, without interruption by other tasks. The two approaches have been implemented in order to be able to compare two SW scheduling algorithms, thus they are both used for scheduling.

For both algorithms, we use a light communication model based on communication time. Communication time between tasks executed in the same unit is assumed to be zero. The transfer of data over the connecting bus is associated with a certain communication time based the the amount of data transferred, the bus width, and the bus frequency.

The extended DLS algorithm has been selected due to its ability to handle heterogeneous multiprocessing architectures consisting of several HW and SW units taking interprocessor communication costs into account. Heterogeneity is represented by varying execution times of tasks, which are included in the Dynamic Level (DL) computation. If a task-processor combination is invalid, its execution time is infinity, leading to a DL of minus infinity. This prevents that combination from being selected. The state of the communication resources are modeled as occupied slots of communication. The state is included in two steps of the algorithm:

- **DL computation:** If the communication resource is free to provide communication from the predecessor to the current node, the communication time is assumed to take place right after finishing the predecessor, else the communication is moved to the next free communication slot. Both possibilities influence the Data Available (DA) time, thus the computation of DL.
- **Scheduling:** When the node with the highest DL is scheduled, it is performed based on the calculated start time in the previous step. This is followed by an update of the state of the communication resources.

TABLE I
DESCRIPTION OF TASK-GRAPHS FOR THE EXPERIMENTS. CP DENOTES THE LENGTH OF THE CRITICAL PATH IN TERMS ON NUMBER OF NODES.

Experiment	Tasks	Edges/Task	CP [nodes]
1	5	1.2	3
2	5	1	4
3	10	1.6	3
4	10	0.8	4
5	10	1.2	5
6	10	1.8	5
7	15	0.8	5
8	15	1	8
9	15	1.2	6
10	15	1.53	6

III. MAPPING EXPERIMENTS

Several mapping experiments have been performed during the development of the framework, they are explained in this section. The experiments were performed as a series of mapping experiments for various task-graphs. The task-graphs had the number of nodes $\{5, 10, 15\}$, with varying numbers of edges and length of the Critical Path (CP). The graphs are described in table I. All graphs have only a single sink node.

TABLE II
ALGORITHM OPTIONS FOR THE MAPPING EXPERIMENTS

No	Temporal Partitioning	Multiprocessor Scheduling
1	Level-based	Simple list-based
2	Clustering-based	Simple list-based
3	Level-based	Extended DLS
4	Clustering-based	Extended DLS
5	0-1 ILP-based Optimal Reference	

The architecture for all experiments was the same, a HW/SW architecture consisting of one SW processor and one HW unit. The HW unit had 15 logic slices, and the reconfiguration time was 10 cycles. Reconfiguration was assumed not to overlap with HW execution, but has no influence on the SW execution. We assumed a constant transfer time of two cycles between the SW and HW units. This transfer was assumed not to interrupt HW nor SW execution.

The SW and HW execution times as well as the HW-cost were randomly created to each task, based on random distributions in the given intervals.:

- SW execution time: $[1; 20]$
- HW execution time: $[1; 10]$
- HW Cost: $[1; 15]$

The experiments were performed for four combinations of our mapping framework as well as the optimal 0-1 ILP reference as indicated in table II. The ILP problem formulation is outlined in the next section III-A. The results were compared in terms of makespan (defined as the total execution time of the task-set) and the exploration time (defined as the execution time of the exploration algorithm). The mapping framework was executed in Matlab® on a standard PC.

A. ILP Formulation of Optimal Mapping

The optimal mapping reference is performed by an 0-1 ILP formulation of the problem. The formulation is a light version of the work by Banerjee et al. [4] and is described below. The major difference between their work and our work is that we only consider the area and have disregarded HW placement constraints that Banerjee et al. use to make sure that tasks that span several columns are placed in consecutive columns. Furthermore, we have added the precedence constraint for reconfiguration in equation (4), such that a HW area is reconfigured before its tasks are executed. The formulation of the problem allows partial reconfiguration, thus potentially a lower makespan than for the global reconfiguration case. First some binary variables are described, indexed by i as the task-index, $i \in \{0, \ldots, n_{\text{tasks}} - 1\}$, and j as the time-step, $j \in \{0, \ldots, n_{\text{timesteps}} - 1\}$. The variables are:

- $x_{i,j}$ is 1 if task T_i starts execution in HW at timestep j, 0 otherwise.
- $y_{i,j}$ is 1 if task T_i starts execution on the SW processor at timestep j, 0 otherwise.
- $r_{i,j}$ is 1 if the reconfiguration for task T_i starts execution at timestep j, 0 otherwise.
- in_{i_1,i_2} is 1 if the communication along the edge between task T_{i_1} and T_{i_2} incurs a communication delay, 0 otherwise.

Furthermore, the costs are given by the symbols:

- t_i^{sw} is the SW execution time of task T_i.
- t_i^{hw} is the HW execution time of task T_i.
- c_i^{hw} is the HW resource cost of task T_i.
- $t_{\text{reconf}}^{\text{hw}}$ is the time is takes to reconfigure the HW.
- C_{FPGA} is the full logic capacity in terms of CLB logic slices of the FPGA.
- ct_{i_1,i_2} is bus data transfer time from task T_{i_1} to T_{i_2}.

The variables are subject to a series of constraints:

1) Uniqueness Constraint: Every task executes only once:

$$\forall i, \sum_j (x_{i,j} + y_{i,j}) = 1 \tag{1}$$

2) SW Processing Constraint: At each time, at most one task is executing on the SW processor:

$$\forall j, \sum_i \sum_{m=j-t_i^{\text{sw}}+1}^{j} y_{i,m} \leq 1 \quad , \tag{2}$$

where the sum over m is performed to include $y_{i,m}$ over all time-steps where a SW task can occupy the SW processor.

3) Reconfiguration Constraint: For each task, there is at most one configuration, expressed as mutual exclusiveness of SW execution and reconfiguration:

$$\forall i, \sum_j (y_{i,j} + r_{i,j}) \leq 1 \tag{3}$$

Furthermore, if the task is performed in HW, reconfiguration must precede execution:

$$\forall i, \sum_j j \cdot r_{i,j} + \sum_j t_{\text{reconf}}^{\text{hw}} \cdot r_{i,j} - \sum_j j \cdot x_{i,j} \leq 0 \tag{4}$$

4) FPGA Resource Constraint: For the FPGA, the sum of resources used for execution or reconfiguration at any timestep must not exceed the full size of the FPGA. A sum over m is included similarly to (2):

$$\forall j, \sum_i \Big(\sum_{m=j-t_i^{\text{hw}}+1}^{j} c_{\text{i}}^{\text{hw}} \cdot x_{i,m} + \sum_{m=j-t_{\text{reconf}}^{\text{hw}}+1}^{j} c_{\text{i}}^{\text{hw}} \cdot r_{i,m} \Big) \leq C_{\text{FPGA}} \tag{5}$$

5) Communication Constraint: Communication on the bus should only be performed when tasks connected by edges are performed on different units:

$$\forall \text{edges}(i_1, i_2), \sum_j y_{i_1,j} + y_{i_2,j} + in_{i_1,i_2} = \{0, 1\} \tag{6}$$

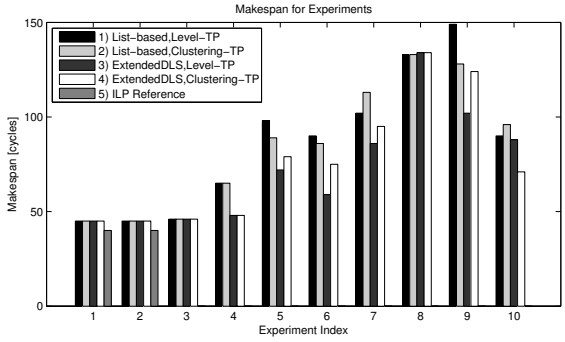

Fig. 4. Results in terms of makespan. The bars 1-4 for each experiment are for the proposed framework. Bar 5 is an adapted version of [4]. In 90% of the cases, the Extended DLS algorithm gave better or equally good results compared to the list-based scheduling. Out of those cases, 44% showed additional improvement in makespan by using the level-based temporal partitioning.

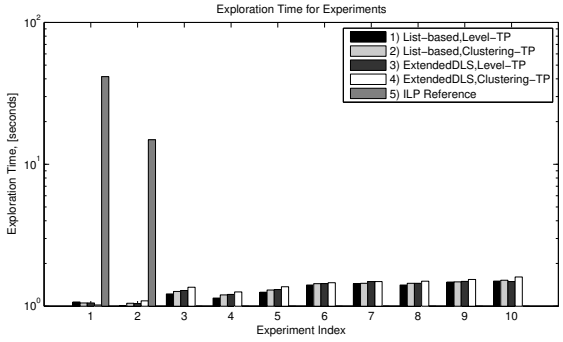

Fig. 5. Resulting in terms of exploration time. The bars 1-4 for each experiment are for the proposed framework. Bar 5 is an adapted version of [4], described in section III-A. The 0-1 ILP solution was only obtained for 20% of the cases, while it had prohibitively long exploration time for the rest of the cases. The results clearly showed that the 0-1 ILP solution is not a viable alternative, whereas the variation in exploration times in the four options of the proposed mapping framework was insignificant.

6) Precedence Constraint:

$$\forall \text{edges}(i_1, i_2), \sum_j \big(j \cdot x_{i_1,j} + j \cdot y_{i_1,j}\big) + \quad (7)$$

$$\sum_j \big(t^{\text{hw}}_{i_1} \cdot x_{i_1,j} + t^{\text{sw}}_{i_1} \cdot y_{i_1,j}\big) + \quad (8)$$

$$ct_{i_1,i_2} \cdot in_{i_1,i_2} - \sum_j \big(j \cdot x_{i_2,j} + j \cdot y_{i_2,j}\big) \quad \leq \quad 0 \quad (9)$$

The optimization goal is given by minimization of the finish-time of the last task, which can be formulated as:

$$\min \sum_j \big(j \cdot x_{n,j} + j \cdot y_{n,j} + t^{\text{hw}}_i \cdot x_{n,j} + t^{\text{sw}}_i \cdot y_{n,j}\big) \quad , \quad (10)$$

where n is the index of the last task (sink node).

Having the ILP-problem defined, it was passed to the solver, `glpsol` version 4.35, from the GNU Linear Programming Kit (GLPK) [8]. The `glpsol` was executed on a standard Linux PC. The results were compared to the result of the mapping framework as described in the previous section III.

IV. Results

The results of the mapping experiments are given by makespan and exploration times shown in the figures 4 and 5, respectively. For the cases where ILP experiments have been performed, the results are shown in the graphs as a rightmost grey bar for each task graph. The optimal ILP solution was only found for 20% of the task-graphs, as the exploration time was simply too long, going beyond more than eight hours for even relatively simple task-graphs with only 10 nodes.

Furthermore, we have included the resulting schedule for task-graph 6 in figure 6, as an illustration of the outcome of the mapping framework.

V. Discussion

When comparing the results presented in figure 5 it is clear that the ILP reference has a significantly higher exploration time than the framework that we propose in section II of this paper. However, when looking at the makespan results in figure 4, the mapping framework resulted in a slightly higher (12.5%) makespan for the experiments 1 and 2. However, the lower makespan of the optimal reference was made possible due to overlaying of HW execution and reconfiguration in dynamic partial reconfiguration.

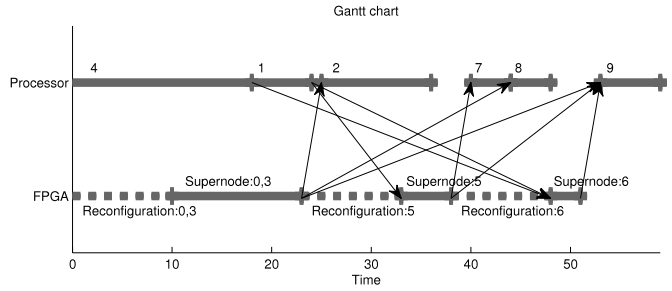

Fig. 6. Resulting schedule of task-graph 6, obtained by level-based temporal partitioning and the Extended DLS scheduling. The dotted lines indicate reconfiguration of the HW, and the arrows represent data transfer on the bus.

When the results are compared for the four different combinations for the presented mapping framework, the results were less clear. For 9 of the 10 cases, the Extended DLS algorithm gave better or equally good results compared to the list-based scheduling. Out of those 9 cases, 4 of them showed that the level-based temporal partitioning gave better results than the clustering-based. Only in 1 of those 9 cases, the level-based performed worse than the clustering-based temporal partitioning. This was surprising since the level-based algorithm would normally lead to more connections to outside partitions, which could potentially increase the HW/SW communication delay. However, the level-based algorithms are less likely to create paths from output to input of the same partition that go through other partitions, thus leading to fewer partitions than the clustering-based approach.

However, it is beneficial to run all four algorithms and compare the results. Such runs do only take short time as seen in figure 5, but gave a highest-to-lowest makespan reduction between 0% and 34%.

The performance of the proposed mapping framework is highly dependent on the early HW/SW partitioning, and it is therefore relevant to consider if this can be improved. First, the reconfiguration time is included for each HW task, even though it may cover reconfiguration of several tasks in parallel (for HW supernodes). This may be improved by weighting the HW reconfiguration time relative to the logic resource usage. However, the partitioner may then not be aware of the risk that small tasks may still require their own partition as described in section II-C. Second, there has not been incorporated any feedback loop into the partitioning as indicated in figure 1. This may be beneficial especially for the partitioning cases where the HW and SW execution times are close to each other.

VI. Conclusion

In this paper we presented a mapping framework for reconfigurable heterogeneous architectures consisting of a SW processor and a HW unit with global reconfiguration capability. Our main contribution is that the framework has been developed with the explicit goal to be able to handle heterogeneous reconfigurable architectures consisting of multiple HW and SW units. The framework is based on an application and architecture description, related through a cost-library that provides information of implementation alternatives of each task. The mapping framework performs HW/SW partitioning, and uses temporal partition algorithms to create HW partitions that can be handled by a scheduling and binding algorithm for heterogeneous multiprocessor architectures.

Mapping experiments were performed for ten task-graphs, with four combinations of two temporal partitions algorithms and two multiprocessor scheduling algorithms. The results showed that the mapping framework had very short exploration time as compared to the (existing) ILP approach, but that the selection of a specific mapping method (out of the four combinations) had an impact of up to 34% compared to the worst performing method. For 90% of the cases, the Extended DLS algorithm in combination with level-based temporal partitioning had the best performance.

We conclude that the proposed mapping methodology is promising and that it can provide designers with a tool for rapid exploration of scheduling strategies for reconfigurable heterogeneous architectures. In order to further improve the methodology, we will conduct the following as future work: a) improve the HW/SW partitioning algorithm, and b) add a feedback loop from the multiprocessor scheduler. Furthermore, future work will also include experiments that cover architectures consisting of multiple SW and HW units.

References

[1] A. Shoa and S. Shirani, "Run-time reconfigurable systems for digital signal processing applications: a survey," *Journal of VLSI Signal Processing Systems*, vol. 39, no. 3, pp. 213–235, 2005.

[2] M. Kaul and R. Vemuri, "Optimal temporal partitioning and synthesis for reconfigurable architectures," in *Proceedings of the conference on Design, automation and test in Europe*, 1998, pp. 389–397.

[3] K. M. G. Purna and D. Bhatia, "Temporal partitioning and scheduling data flow graphs for reconfigurable computers," *IEEE Trans. Comput.*, vol. 48, no. 6, pp. 579–590, Jun. 1999.

[4] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, "Integrating physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration," *IEEE Trans. VLSI Syst.*, vol. 14, no. 11, pp. 1189–1202, Nov. 2006.

[5] J. Noguera and R. M. Badia, "A hw/sw partitioning algorithm for dynamically reconfigurable architectures," in *Proceeding of Design, Automation and Test in Europe*, Mar. 2001, pp. 729–734.

[6] K. S. Chatha and R. Vemuri, "Hardware-software codesign for dynamically reconfigurable architectures," in *9th International Workshop on Field-Programmable Logic and Applications*, 1999, pp. 175–184.

[7] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

[8] GNU, "Gnu linear programming kit (glpk)," http://www.gnu.org/software/glpk/.