Evaluation of Runtime Task Mapping Heuristics with rSesame - A Case Study

Kamana Sigdel[†] Mark Thompson[‡] Carlo Galuzzi[‡] Andy D. Pimentel[†] Koen Bertels[†]

[†] Computer Engineering Laboratory
EEMCS, Delft University of Technology, The Netherlands
Email: {K.Sigdel, C.Galuzzi, K.L.M.Bertels}@tudelft.nl

[‡] Computer Systems Architecture Group
University of Amsterdam, The Netherlands
Email: {M.Thompson, A.D.Pimentel}@uva.nl

***Abstract*—rSesame is a generic modeling and simulation framework which can explore and evaluate reconfigurable systems at the early design stages. The framework can be used to explore different HW/SW partitionings, task mappings and scheduling strategies at both design time and runtime. The framework strives for a high degree of *flexibility*, *ease of use*, *fast performance* and *applicability*. In this paper, we want to evaluate the framework's characteristics by showing that it can easily and quickly model, simulate and compare a wide range of runtime mapping heuristics from various domains. A case study with a Motion-JPEG (MJPEG) application demonstrates that the presented model can be efficiently used to model and simulate a wide variety of mapping heuristics as well as to perform runtime exploration of various non-functional design parameters such as execution time, number of reconfigurations, area usage, etc.**

I. Introduction

Recent trends show that modeling and simulation frameworks are becoming one of the popular ways for exploring and evaluating reconfigurable systems [1-5]. These frameworks can explore and evaluate reconfigurable systems' behavior at various design stages. The advantages introduced by such approaches include high model (component) re-usability and easy customization of the design according to various system requirements. Nevertheless, most of the available simulation frameworks are restricted to design time exploration and only deal with the static nature of the reconfigurable architectures and/or the applications. Design time exploration alone is not adequate and cannot address the dynamic nature of such achitectures/applications. As a consequence, to cope with this situation, researchers build their own proprietary simulation tools to model and evaluate reconfigurable architectures at runtime. To this end, there are several issues: 1) the lack of standardized methods of runtime modeling and simulation for evaluating such architectures, 2) the complexity of the evaluation procedure and 3)the cumbersomeness of the comparison between the various evaluation methods. To address these challenges, there is a need for a standard modeling and simulation tool for Design Space Exploration (DSE) which is able to explore and evaluate reconfigurable systems' behavior at runtime. By having such a framework, it is possible to evaluate reconfigurable systems at runtime and, more important, to provide a common platform for model comparison and standardization of the benchmarking. Towards this goal, in [10], we presented a generic modeling and simulation framework, rSesame, which can explore and evaluate reconfigurable systems at early design stages. The framework can be used to perform DSE with respect to HW/SW partitioning, task mapping and scheduling at both design time and runtime. With rSesame, designers can instantiate a model that can explore and evaluate any kind of reconfigurable architecture running any set of streaming applications from the multimedia domain. The framework provides various important system attributes such as execution time, area usage, number of reconfigurations, etc and it thrives for a high degree of *flexibility*, *ease of use*, *fast performance*, and *applicability*.

In this paper, we evaluate the framework's characteristics by illustrating that it can easily and quickly model, simulate and compare a wide range of runtime task mapping heuristics from various domains. To demonstrate this, we instantiate a model from the framework for the Molen architecture [11]. We incorporate several task mapping heuristics with the model and we evaluate these heuristics based on various non-functional attributes recorded from the model. The main contributions of this paper are the following:

- evaluation of the rSesame framework with the model instantiated for the Molen reconfigurable architecture;
- a case study of three different runtime mapping heuristics with the model
- evaluation and comparison of the aforementioned heuristics for the first time using a single common modeling and simulation platform.

II. Related Work

In the traditional way of performing DSE, various algorithms of different complexity are used for HW/SW partitioning, task mapping and scheduling. Examples of such classical algorithms are dynamic programming, branch and bound, integer linear programming, graph partitioning, simulated annealing, genetic algorithms, ant colony optimization, etc. Besides these algorithmic approaches, various models are also used for DSE to evaluate reconfigurable systems behavior at various design stages. In [2], the authors present a modeling methodology for runtime scheduling of reconfigurable architectures based on discrete event systems. In [3] and [4], the authors present a system-level modeling framework for the rapid exploration of different reconfiguration alternatives. Another approach for simulating the performance of reconfigurable architectures based on SystemC has been presented in [1]. Similarly, in [7], a methodology for modeling of dynamically re-configurable blocks at the system-level using SystemC is presented. However,

[0]This research has been funded by the hArtes project EU-IST-035143, the Morpheus project EU-IST- 027342 and the Rcosy Progress project DES-6392.

all these approaches are limited to design time exploration and only deal with the static nature of the architecture and/or the applications. To address the dynamic nature of dynamic reconfigurable systems, decisions made only at design time are not adequate and cannot address all runtime system conditions. There are few attempts which combine design time exploration together with runtime management, presented in [5] [6] and try to evaluate the system at both stages. However, these methodologies are mostly restricted to the MPSoC domain and do not address the reconfigurable system domain. Unlike existing approaches, we focus on designing a system-level modeling and simulation framework for the exploration and the evaluation of reconfigurable architectures at early design stages. We make a first attempt to present a generic framework which can evaluate reconfigurable systems at *design time* as well as at *runtime*.

III. rSESAME OVERVIEW

The rSesame framework is built upon the Sesame framework [9]. Sesame is a modeling and simulation platform for system level DSE targeting streaming applications from the multimedia domain. The rSesame is a reconfigurable extension to Sesame, which can be efficiently employed to perform DSE for HW/SW partitioning, task mapping and scheduling at design time as well as at runtime for any reconfigurable systems. The framework strives for several key features such as *flexibility*, *ease of use*, *fast performance* and *applicability*. The rSesame framework allows an application task to be modeled either as a HW, SW or as a *pageable* task. A HW/SW task is always mapped onto the reconfigurable hardware component/microprocessor, while a *pageable* task can be mapped on either of these resources. Task assignment to the SW, HW and pageable categories is done at design time. At runtime, these tasks are mapped onto their corresponding resources based on time, resources and conditions of the system.

We use Kahn Process Network (KPN) [12] at the granularity of coarse-grain tasks for application modeling. Each KPN process contains functional application code instrumented with annotations that generate Read, Write and Execute events describing the actions of the process. The generated traces are forwarded onto the architecture layer using an intermediate mapping layer which consists of Virtual Processors (VPs) to schedule these traces. Along with the VPs, the mapping layer contains a Runtime Mapping Manager (RMM) that deals with the runtime mapping of the applications on the architecture. Depending on current system conditions, the RMM decides where and when to forward these events. To support its decision making, the RMM employs an arbitrary set of user-defined policies for runtime mapping which can simply be plugged in and out of the RMM. The RMM also collaborates with other architectural components to gather architectural information. The architecture layer in the framework models the architectural resources and constraints. These architectural components are constructed from generic building blocks provided by a library, which contains components for processors, memories, on-chip network components, etc. As a result, any kind of reconfigurable architecture can be constructed from these generic components. Besides the regular parameters such as computation and communication delays, other architectural parameters like reconfiguration delay and area for the reconfigurable architecture, can be provided as extra information to these components. More information on the rSesame framework can be found in [10].

IV. CASE STUDY

In this section, we describe a case study to show the characteristics of the rSesame framework tested on a real reconfigurable architecture by evaluating and comparing different runtime mapping heuristics based on various design attributes.

A. Model Instantiation

In order to carry out the evaluation, we instantiate a model from the rSesame framework for the Molen reconfigurable architecture. The rSesame framework is not restricted to a specific kind of architectures and it can be deployed to evaluate any reconfigurable architecture. The Molen architecture is just considered as an example of such an instance. The Molen [11] is an established norm for the polymorphic processor paradigm incorporating a General Purpose Processor (GPP) and a Reconfigurable Processor (RP) such as an FPGA. The RP is used to accelerate code fragments from applications in a processor/co-processor fashion. The RP consists of one or more Custom Computing Units (CCUs), each representing a hardware implementation of a task. Application tasks can be executed either on the GPP (as regular compiled microprocessor code) or on the RP (as a hardware IP core) or on both.

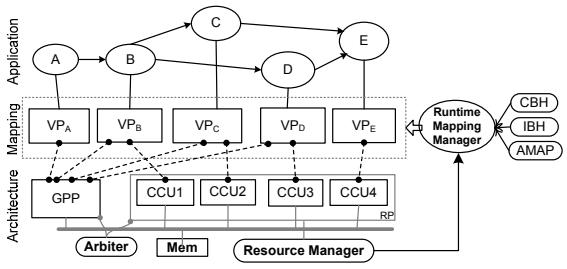


Fig. 1. A Model instantiated from the rSesame framework that can facilitate runtime task mapping for the Molen reconfigurable architecture

Fig 1 depicts the model instantiation that can perform runtime task mapping for the Molen architecture. In the model, CCUs and the GPP are modeled as architectural layer components. A Resource Manager (RM) is added to monitor which CCUs are *configured* and to keep track of the architectural resource information (e.g. available area). The RMM collaborates with the RM to gather architectural information such as free resources. The arbiter is modeled as an architectural component and performs synchronization between the GPP and the RP, which supports either mutual exclusive operation of GPP and RP (traditional co-processor model) or parallel operation.

B. Experimental Setup

We consider a Motion-JPEG (MJPEG) encoder application as a case study. The corresponding KPN graph is shown in Fig 2. The application model consists of two implementations (MJPEG1 and MJPEG2) of the MJPEG application. MJPEG1 operates on the blocks (partially) in parallel (see the 4 DCT/Q streams in the top part of Fig. 2), whereas MJPEG2 operates on the blocks sequentially (bottom part of Fig. 2). MJPEG1 and MJPEG2 are combined together in order to create an example of dynamic application. MJPEG2 can be considered as a sporadic application that appears in the system randomly and competes

with MJPEG1 for the resources. This behavior is implemented in such a way that at a certain point in time MJPEG2 starts encoding a frame simultaneously with MJPEG1.

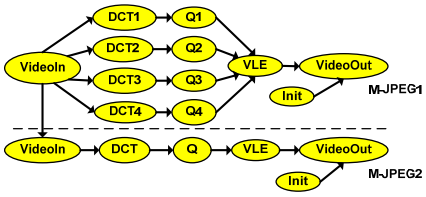


Fig. 2. Application Model

We created a Molen architecture model with 18 CCUs, one for each task. Note that the number of CCUs is a parameter that a designer can define based on the number of pageable and HW tasks. For this experiment, we consider all tasks as pageable to fully exploit runtime mapping by deciding *where* and *when* to map them at runtime. We assume that no task can have a size larger than the total FPGA area. All CCUs may not fit on the RP at once because of area constraints. Nonetheless, they can execute on the RP after the reconfiguration. We use estimated values of computational latency (for the GPP and the CCUs), area occupancy (on the FPGA) and reconfiguration delay for each task. No delay is associated with the RMM, the RM or with context switching. The main purpose of this case study is to evaluate the framework's characteristics with the use of the instantiated model. Hence, the calibration of the model is left as a future work.

C. Task Mapping Heuristics

Our framework allows easy modification of certain components in the model, while keeping other parts untouched. We illustrate this feature by allowing the designers to experiment with different kinds of runtime application mapping heuristics. The heuristics under consideration have variable complexity with respect to their implementation and the nature of their execution. They were originally defined to be used at different system stages ranging from lower architecture level to OS and higher application levels. This illustrates the *flexibility* of the framework in incorporating different kinds of algorithms from various domains. In the following, we describe the studied heuristics more in detail:

As Much As Possible Heuristic (AMAP): AMAP tries to maximize the use of FPGA area as much as possible [8]. Tasks are mapped onto the RP if area is available, otherwise they are mapped on the GPP. This simple heuristic is being used in resource management in various domains. The implementation of this heuristic is trivial within our model and can be described using only a few lines of code.

Cumulative Benefit Heuristic (CBH): CBH maintains a cumulative benefit (CB) value for each task that represents the amount of time that would have been saved up to that point if the task had always been executed on the RP. Mapping decisions are made based on these values and the available area. For example, if the available area is not sufficient to load the CCU for the current task, CCUs can be swapped if the CB of the current task is higher than that of the to-be-swapped-out set. In [13], this heuristic is used for dynamic coprocessor management of reconfigurable architectures at a low architecture level. The implementation of this heuristic was a little more complicated and resulted with more lines of code than the one of AMAP.

Interval Based Heuristic (IBH): IBH divides execution into a sequence of time slices (intervals) for mapping and scheduling. In each interval, execution frequency of each task is counted. Mapping decisions are made based on the frequency count of the previous interval: tasks with the highest frequency count are mapped onto the RP. In [14], this heuristic is used for resource management in a multi-threaded environment at OS level. To implement this heuristic, intervals are marked by inserting a special event in the application model, which is a trivial process within Sesame. Whenever this special event is encountered in the mapping layer, the frequency count is revisited and the task mapping is changed. We chose intervals to coincide with frame boundaries, although they can easily be defined in different ways.

V. Results and Evaluation

The model provides various useful statistics to the designer, such as total execution time (in terms of simulated cycles), area usage, number of reconfigurations, etc. It also provides various runtime information about the application and the architecture. By observing these statistics, it is possible to gain useful insight into the characteristics of the architecture and the efficiency of the mapping heuristics. In the following, we describe these statistics in more detail.

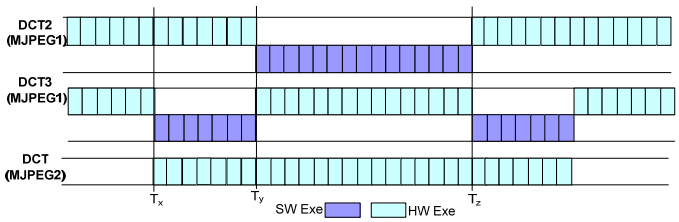


Fig. 3. DCT Execution snapshot in MJPEG1/MJPEG2 (AMAP)

A. Runtime Information

This information is recorded as a trace during the model execution. Currently, the model is capable of providing the runtime analysis described in the following:

Spatial behavior of a task - The spatial behavior of a task provides an indication whether a pageable task is running as a HW or SW task. This information is vital to check the correctness of the spatial mapping behavior. Fig 3 captures a snapshot of such behavior for three different tasks - DCT2, DCT3 (from MJPEG1) and DCT (from MJPEG2) recorded for the AMAP heuristic. The figure shows that in order to accommodate DCT on the RP, DCT2 (at time T_y) and DCT3 (at times T_x and T_z) switched their mapping to SW providing DCT enough area to execute. Such mapping behavior for all the tasks for the entire execution time-line can be retrieved from the model.

Temporal behavior of a task - A HW task can further show various behaviors depending on its execution. It can either be in a waiting state, a mapped state, or running state. A HW task is in a waiting state if the task is waiting to be mapped onto the RP. This happens when there is no area available on the RP or in case of a task dependency with other tasks. A HW task is in a mapped state if it is already configured on the RP and it is not currently executing, however, it may execute again. A

HW task is in the running state when the task is actually busy performing execution.

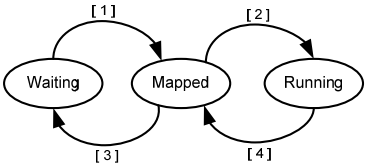


Fig. 4. A FSM showing temporal behavior of a HW task

Fig 4 presents a finite state machine (FSM) showing different states of a HW task, where the numbers 1 to 4 refer to the following state transitions: 1) as soon as area becomes available or task dependency ends, 2) the task execution starts, 3) when other tasks need to be executed, and 4) the task execution finishes but the task may execute again. The mapped state has a reconfiguration delay associated with it. If a task transits from a waiting state to a running state, this delay is considered but if the task is already in the mapped state then this delay is ignored. The performance can be significantly improved by avoiding the former transition. A HW task may or may not enter the waiting state depending on the system conditions. To avoid a task to enter the waiting state due to a lack of area on the RP, the task can be mapped onto the GPP. This decision is made by the specific policy implemented by the RMM and/or by the RM. However, if the waiting state is due to a data dependency, it cannot be avoided. Table I shows a snapshot of the temporal behavior of each HW task during a small period of the application run recorded for the AMAP heuristic. At each execution, the behavior of each HW task is noted as R, M and W which refer to the **R**unning, **M**apped and **W**aiting state respectively. In each row, the state of all the HW tasks is recorded at each execution. As it can be inferred from the table, HW tasks change their state (R, M and W) with time as per required by the system. The first row shows that DCT2 and DCT3 are in the running state, DCT1, DCT4 and VideoOut1 (VOut1) are in the mapped state while the other tasks are in the waiting state. The RP has area limitation and as a result, only five tasks can be in the mapped/running state. Moreover, all the Q tasks, VLE1 and VOut1 have a data dependency with DCTs. As a result, other tasks are in the waiting state. In the successive executions, these tasks, in turn, are mapped and run.

Similarly, when DCT4 changes its state from M to R, the reconfiguration is avoided. However, in the case of DCT1, when the state changes from W to R (as it has to pass through mapped state), the reconfiguration delay is added. In the latter case, by saving the first M state (see first row for DCT1 in Table I) for three more executions, this delay can be easily avoided. The mappings can be optimized by understanding and analyzing such behavior. Thus, this information is vital not only to test the correctness of the mapping algorithms but also for their optimization.

Spatial behavior of tasks can also be observed from the table. For example, when MJPEG2 arrives, VideoIn1(VIn1) from MJPEG1 is moved to the GPP (indicated by SW in the table) and DCT from MJPEG2 is mapped onto the RP. This is again due to the area limitation on the RP.

Number of hardware or software tasks - It gives information about the total number of tasks being executed on HW and SW at a particular time during the simulation run. Fig 5 shows

TABLE I
A SNAPSHOT OF THE TEMPORAL BEHAVIOR OF HW TASKS [R = RUNNING, W = WAITING AND M = MAPPED STATE] (AMAP)

Time

VIn1	DCT1	DCT2	DCT3	DCT4	Q1	Q2	Q3	Q4	VLE1	VOut1	DCT
...											
W	M	R	R	M	W	W	W	W	W	M	-
R	W	R	R	M	W	W	W	W	W	M	-
M	W	R	R	R	W	W	W	W	W	M	-
R	W	R	R	R	W	W	W	W	W	M	-
M	R	R	R	R	W	W	W	W	W	W	-
R	R	R	R	R	W	W	W	W	W	W	-
- - - - - - - - - - - MJPEG2 starts - - - - - - - - - - -											
SW	R	R	M	R	W	W	R	W	W	W	W
SW	R	R	W	R	W	W	R	W	W	W	R
SW	R	W	W	R	W	R	R	W	W	W	R
SW	R	W	W	W	W	R	R	R	W	W	R
SW	R	W	W	W	W	M	W	R	R	W	R
SW	R	W	W	W	W	W	W	M	M	R	M
SW	R	W	W	W	W	W	W	M	R	M	M
SW	R	W	W	W	W	W	W	M	M	R	M
...											

this information for all three heuristics at various checkpoints of the execution time-line. For CBH, at the first checkpoint, only MJPEG1 is running and DCT1, DCT2, DCT3, DCT4 and Q2 are mapped onto the RP. At the second checkpoint (see bi-direction arrow in the figure), when MJPEG2 arrives, Q2 is pushed to the GPP and DCT from MJPEG2 is mapped onto the RP. CBH maps the tasks with largest CB value onto the RP. In this case, the CB of DCT is larger than the CB of Q2. As a result, Q2 is swapped with DCT for execution. In the case of AMAP, different task sets are mapped onto the RP than in the case of CBH. This task set also changes after the arrival of MJPEG2. AMAP changes the application mapping more frequently than CBH and it can map any task onto the RP. As a result, in the specified period in Fig 5(b), these tasks are accumulated. The detailed representation of a quarter of a period in Fig 5(b) is given as a snapshot in Fig 3. Similarly, IBH determines the task sets for HW mapping at the beginning of each interval and continues mapping the same task set during the whole interval. As it can be inferred from Fig 5(c), in the first checkpoint only tasks from MJPEG1 are mapped onto the RP. This task set changes in the second checkpoint after the arrival of MJPEG2 and stays the same further on. All the three diagrams show the accumulated tasks for each interval. Thus, in case of AMAP and CBH, these tasks may be different in each snapshot within one interval. However, in case of IBH, the task set stays the same in each snapshot within one interval. The detailed expansion of Fig 5 will take the form of Fig 3. The information provided by these figures are indispensable in order to evaluate the correctness of the heuristics.

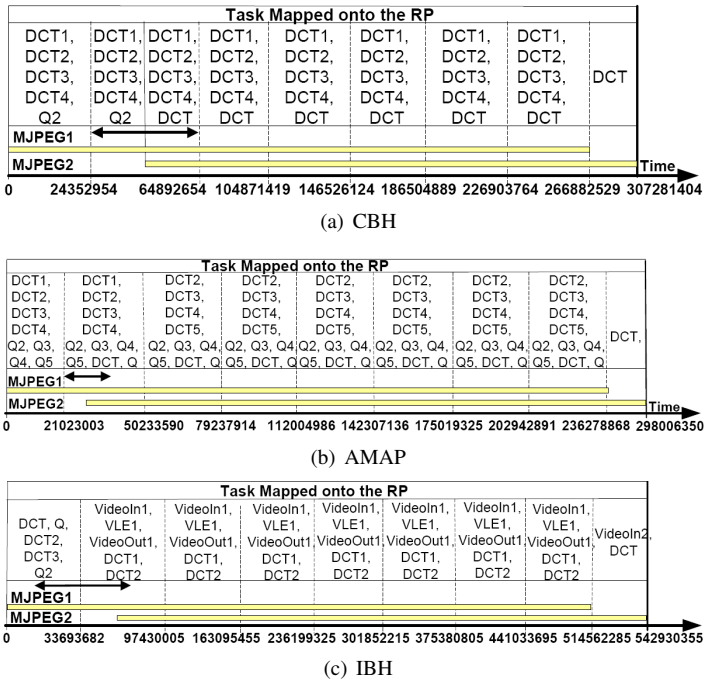
For this case study, we note that all the above system-level simulations (with the inputs for MJPEG1 and MJPEG2 consisting of 8 and 4 picture frames of 128×128 pixels respectively) can be executed in less than 10 second, thus allowing *fast performance* of the model and extensive design space exploration.

B. Execution Time

The execution time is recorded in terms of simulated clock cycles. The SW execution time is noted as the total number of cycles when all the tasks are mapped onto the GPP only and HW execution time is recorded when tasks are mapped onto the RP. The speed-up is calculated as a ratio of these two values.

C. Percentage of HW/SW Execution Time

The percentage of HW execution and SW execution is computed as the total percentage of the execution time contributed



(a) CBH

(b) AMAP

(c) IBH

Fig. 5. Task mapping breakdown with respect to time for different Heuristics

by the RP for HW execution and the total percentage of the execution time contributed by the GPP for SW execution of an application respectively. Similarly, the percentage of reconfiguration time calculates the percentage of the total execution time spent in reconfiguration. This provides an indication on how much of the total time is spent in the computation and how much is just spent in reconfigurations. These values are calculated as follows:

$$\text{SW Exec.}(\%) = \frac{\sum_{i=1}^{N} \#\text{SWMappings}(T_i) \cdot \text{SWExecTime}(T_i)}{\text{TotalExecTime}} \cdot 100 \quad (1)$$

where $\#\text{SWMappings}(T_i)$ is the total number of SW mappings counted by the model for task T_i, $\text{SWExecTime}(T_i)$ is the software execution latency for task T_i and TotalExecTime is the total simulated execution time. HW execution time is given as:

$$\text{HW Exec.}(\%) \leq \frac{\sum_{i=1}^{N} \#\text{HWMappings}(T_i) \cdot \text{HWExecTime}(T_i)}{\text{TotalExecTime}} \cdot 100 \quad (2)$$

Note that the HW execution percentage can only be given here as an upper bound, since the execution of tasks on the RP can be performed in parallel. The metric calculated here is an accumulated value. The simulator however can give the actual snapshots. A similar equation holds for the time spent reconfiguring:

$$\text{Reconfig.}(\%) \leq \frac{\sum_{i=1}^{N} \#\text{Reconfig}(T_i) \cdot \text{ReconfigDelay}(T_i)}{\text{TotalExecTime}} \cdot 100 \quad (3)$$

where $\#\text{Reconfig}(T_i)$ is the number of times T_i is configured and $\text{ReconfigDelay}(T_i)$ is the reconfiguration delay of T_i.

D. Number of Reconfigurations

The number of reconfigurations is recorded as the total number of reconfigurations incurred during the execution of an application onto a given architecture. This provides an indication on how efficiently the reconfiguration delay was avoided while mapping tasks onto the RP. For example, mapping task A, task B, and then task A again on the RP requires 3 reconfigurations, while changing this sequence of mapping to task A, task A and then task B requires only 2 reconfigurations.

E. Time Weighted Area Usage

The weighted area usage factor is a metric that computes how much area is used throughout the entire execution of an application on a particular architecture. This provides an indication on how efficiently the RP area is utilized. This is calculated as follows:

$$\text{Area Usage}(\%) = \frac{\sum_{i=1}^{N} \text{Area}(T_i) \cdot \text{HWExecTime}(T_i) \cdot \#\text{HWMappings}(T_i)}{\text{TotalExecTime} \cdot \text{Area(RP)}} \cdot 100 \quad (4)$$

where $\text{Area}(T_i)$ is the area occupied by task T_i on the RP, $\text{HWExecTime}(T_i)$ is the hardware execution latency of T_i, $\#\text{HWMappings}(T_i)$ is the total number of HW mappings counted by the model for task T_i, Area(RP) is the total area available on the RP and TotalExecTime is the total execution time of the application.

F. Heuristics Comparison

In this section, we compare the three heuristics AMAP, CBH and IBH by using the information recorded from the model as described in the previous subsection. Fig 6 depicts the results of running the three heuristics for mapping the MJPEG application onto the Molen architecture. Interestingly enough, it can be seen that AMAP achieves performance close to CBH in terms of area usage, number of reconfigurations and the total execution time. We might expect a significant difference since AMAP maps random tasks on the RP, whereas CBH maps carefully selected tasks onto the RP.

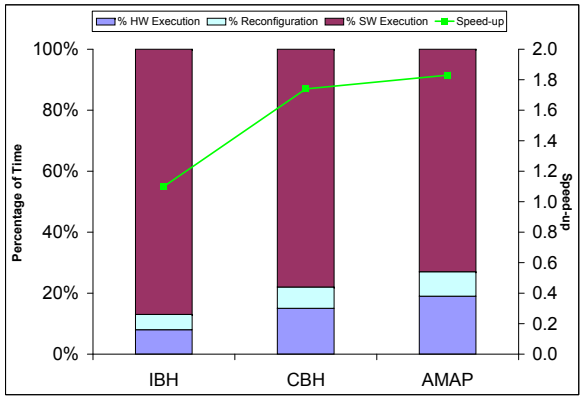
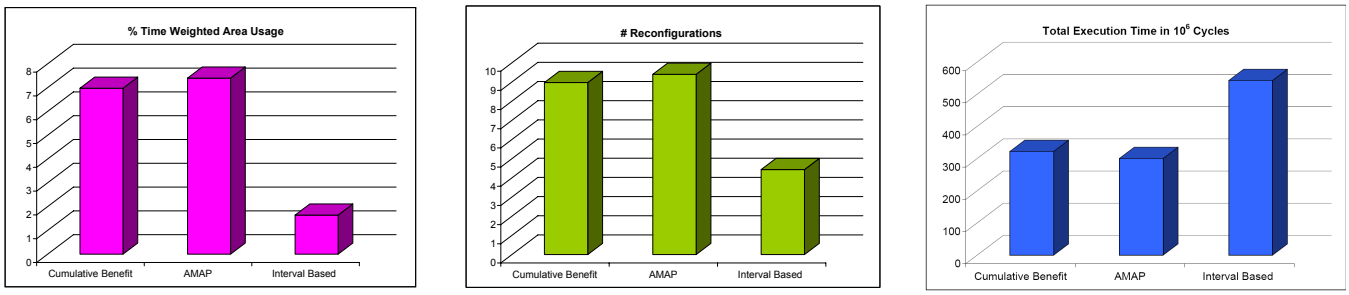


Fig. 7. Heuristics Comparison in terms of HW Execution, SW Execution and Reconfiguration with corresponding Application Speed-Up

As a matter of fact, CBH tries to choose a task with highest HW/SW latency difference and maps it onto the RP if it is more beneficial than executing it on the GPP. However, in this experiment the reconfiguration delay considered for all the



(a) % Time Weighted Area Usage (b) Number of Reconfigurations (c) Total Execution Time

Fig. 6. Comparison of three Heuristics based on Area Usage, Number of Reconfigurations, and Total Execution Time

tasks is a constant factor. As a result, no matter which tasks are mapped onto the RP, the performance of both heuristics is almost the same. Nevertheless, the CBH can perform better in the cases where reconfiguration delay is variable. On the contrary, in the case of IBH, tasks are mapped onto the RP based on the execution count. As a result, many of the sequential tasks (in this case mostly VideoIn and VLE) are mapped onto the RP, thus making inefficient use of the parallelism available in the RP. As a result, this heuristic lags far behind the others. A similar explanation holds for the area usage as well (as shown in Fig 6(a)). Note that the number of reconfigurations for IBH is significantly lower compared to the other two. This is not due to an efficient algorithm which tries to optimize the reconfiguration delay, but this is because the HW mapping count is very low in this case.

Similarly, Fig 7 shows the comparison between the three heuristics in terms of HW and SW execution with respect to the speed-up. The primary y-axis in the graph is stacked as 100% and shows the contribution of HW execution, SW execution and reconfiguration to the total execution time. The secondary y-axis is the measure of the obtained speed-up. Having more HW task mappings, increases the HW execution speed, which in turn accelerates the application. However, it has a penalty to pay with respect to reconfiguration. The efficiency of the mapping heuristics lies in finding the best mapping while minimizing the number of reconfigurations. Nevertheless, in the figure, we see a linear contribution of the reconfiguration overhead to the total execution time. This is again due to the constant reconfiguration delay considered in the experiment. Note that this result can change drastically with a larger diversity of task sizes and reconfiguration delays. In the future, we will perform more experiments with real values of task sizes and reconfiguration delays.

Another observation that can be made from Fig 7 is the contribution of the HW execution, SW execution and reconfiguration to the total execution time. The figure shows that most of the application is executed on the GPP and only less than 30% of the total computation is done on the RP. This is due to the architectural restrictions. Due to the processor/co-processor nature of the studied Molen architecture, the GPP and the RP run in a mutual exclusive way. This influences the mapping decision of the RMM, which in turn contributes to the lower HW execution rates. This significantly increases the total execution time. Note that the area usage is a time weighted factor in terms of total execution time (see equation 4). Thus, these two factors significantly contribute to the low area usage. The area usage can be increased either by mapping more tasks onto the RP or by operating the RP and the GPP in parallel. The case study demonstrates that the framework is flexible and can efficiently assess various runtime mapping heuristics in terms of various design parameters. The comparison shows that the AMAP heuristic performs better in those cases when reconfiguration delay and area are considered as a constant factor. In other cases, CBH may perform better.

VI. Conclusion and Future Work

In this paper, we described and tested a generic modeling and simulation framework for runtime task mapping for reconfigurable architectures. We instantiated a model for the Molen reconfigurable architecture to deploy the framework and used it to explore various design parameters. Due to the fast execution times, the model can be used to efficiently explore and/or evaluate various task mappings and record various architectural parameters such as execution time, area usage, number of reconfigurations and percentage of HW/SW mapping. We showed that the model is easy to construct and extend. This indicates that the presented framework can be efficiently used as a standard platform to facilitate easy comparison between various evaluations and, hence, can also be used as a reference tool for future research. In future work, we will validate the framework against a real implementation for final calibration in order to evaluate and increase its accuracy.

References

[1] Y. Qu et al., "Systemc-based design methodology for reconfigurable system-on-chip", *Proc. of the Euromicro Conf. on DSD05*, 2005.
[2] J. Noguera et al., "System-level power-performance trade-offs in task scheduling for dynamically reconfigurable architectures", *Proc. of CASES03*, 2003.
[3] P.-A. Hsiung et al., "Perfecto: A systemc-based design-space exploration framework for dynamically reconfigurable architectures," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 3, pp. 1–30, 2008.
[4] T. Rissa et al., "System-level modelling and implementation technique for run-time reconfigurable systems" *Proc. of FCCM02*, 2002.
[5] C. Ykman-Couvreur et al., "Design-time application exploration for mp-soc customized run-time management", in *Proc. of SoC05*, 2005, pp. 66–69.
[6] V. Nollet et al., "Run-time management of a mpsoc containing fpga fabric tiles," *IEEE Trans. VLSI System*, vol. 16, no. 1, pp. 24–33, 2008.
[7] A. Pelkonen et al. "System-Level Modeling of Dynamically Reconfigurable Hardware with SystemC", *Proc. of IPDPS03*, USA, 2003.
[8] K. Sigdel et al., "System-level runtime mapping exploration of reconfigurable architectures", *Proc. of RAW09*, 2009.
[9] A. D. Pimentel et al., "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, 2006.
[10] K. Sigdel et al., "rSesame-A generic system-level runtime simulation framework for reconfigurable architectures", *Proc. of FPT09*, 2009.
[11] S. Vassiliadis et al., "The molen polymorphic processor," *IEEE Trans. Comput.*, pp. 1363– 1375, November, 2004.

[12] G. Kahn, "The semantics of a simple language for parallel programming," *Proc. of the IFIP74*, 1974.
[13] C. Huang and F. Vahid, "Dynamic coprocessor management for fpga-enhanced compute platforms," in *Proc. of CASES08*, 2008.
[14] W. Fu and K. Compton, "An execution environment for reconfigurable computing," in *Proc. of FCCM05*, 2005.