Models and Reconfiguration Problems for Multi Task Hyperreconfigurable Architectures*

Sebastian Lange and Martin Middendorf
Department of Computer Science,
University of Leipzig
Augustusplatz 10-11, D-04109 Leipzig, Germany
{langes, middendorf}@informatik.uni-leipzig.de

Abstract

Hyperreconfigurable architectures can adapt their reconfiguration abilities during run time and have been proposed to increase the speed of dynamic reconfiguration. They use two types of dynamic reconfiguration steps. In hyperreconfiguration steps they change their ability for reconfiguration and in ordinary reconfiguration steps they reconfigure the actual contexts for a computation within the limits that have been set by the last hyperreconfiguration step. In this paper we study the concept of partial hyperreconfiguration for multi tasks environments. We propose several models for partially hyperreconfigurable architectures and study corresponding reconfiguration problems to find optimal (hyper)reconfigurations. While under a general cost model the problem to find optimal (hyper)reconfigurations is known to be NP-complete even for a single task we identify an interesting special case that can be solved by polynomial time algorithm even for multiple tasks. We illustrate the introduced concepts with a partially hyperreconfigurable example architecture and describe the results of simulated runs with a small test application.

1 Introduction

The development of dynamically reconfigurable architectures/systems offers new abilities for a flexible and fast execution of computations which change their requirements to the architecture during runtime. Such systems can be reconfigured during runtime to change their internal communication structure and/or their functional units. A problem that arises with increasing abilities for reconfiguration is the growing amount of information that is needed for a reconfiguration operation to define the new state. In addition, the high integration of reconfigurable hardware, e.g. reconfigurable circuits on an FPGA chip, requires large bandwidths for the transfer of reconfiguration information when reconfiguration operations are done at runtime speed. The amount of reconfiguration data that is needed for modern FPGAs for example is several megabytes for a single reconfiguration step. This large amount of necessary information transfer becomes especially time critical for computations that exploit the full capacity of dynamically reconfigurable architectures by frequent reconfigurations.

Some approaches have been proposed in recent years to cope with this problem. Off-line compression methods have been applied to the reconfiguration bit stream before it is loaded onto the system by Dandalis and Prasanna [3]. Additional hardware on the chip allows then to decompress the reconfiguration bit stream during run-time before it is needed to define the next configuration. A compression method that is suitable especially for the reconfiguration bit stream of the Xilinx XC6200 architecture has been described by Hauck et al. [6]. Another approach is to compute the bits which are necessary for reconfiguration directly on chip (see Köster und Teich [8], Sidhu et al. [13], Wadhwa und Dandalis [16]). For Multi FPGA systems it has been proposed not to reconfigure all FPGAs at the same time, but perform the reconfiguration incrementally (Lee and Wong [10]). All these approaches have in common that they do not change the reconfiguration information itself.

A different approach with the aim to reduce the reconfiguration information has been proposed recently by the authors (see [12, 9]). The idea of this approach is to make the potential for reconfiguration itself reconfigurable. This means that the reconfiguration potential of an architecture can be decreased during such periods of a computation where it requires only minor reconfiguration features. During such periods only the new states of the (few) actually

*This work was financed by the German Research Foundation (DFG) through the project "Models and Algorithms for Hyperreconfigurable Architectures" within the priority programme 1148 "Reconfigurable Computing Systems"

available reconfiguration features have to be defined during reconfiguration and therefore the amount of necessary reconfiguration information is small. Two types of reconfiguration steps are used on such architectures: i) reconfiguration steps where the reconfiguration potential of the architecture is defined (*hyperreconfiguration steps*), and ii) ordinary reconfiguration steps which are used to reconfigure the actual context which is used by the computation (simply called reconfiguration steps). Reconfigurable architectures that use these two types of reconfiguration steps are called *hyperreconfigurable architectures*.

Hyperreconfigurable architectures are a promising concept for fast dynamic reconfiguration. Especially for applications where computations typically consist of different phases that use only small parts of the whole reconfiguration potential of the architecture. The smaller the potential of the architecture, that is actually available, as has been defined through a hyperreconfiguration, the less reconfiguration bits are needed for the (ordinary) reconfiguration steps and the faster is such a reconfiguration step.

In this paper we propose several models for partially hyperreconfigurable architectures where several tasks can run in parallel. For such multi task hyperreconfigurable architectures we study the problem of finding optimal (hyper)reconfigurations. While this problem is known to be NP-complete even for a single task under a general cost model we identify an interesting special case that can be solved in polynomial time even for multiple tasks. We give an example for a multi task hyperreconfigurable architecture and show some experimental results for the (hyper)reconfiguration problem on this architecture. Due to space limitations some topics can only be sketched and will be discussed in more detail in the full paper.

2 Hyperreconfigurable Architectures

In this section we describe hyperreconfigurable machines as they have been introduced in [12, 9]. A reconfigurable system is called hyperreconfigurable when its ability for reconfiguration is reconfigurable itself ([12]). Hyperreconfigurable architectures have two types of reconfiguration steps. *Ordinary reconfiguration steps* allow for changing the context of an algorithm during runtime, e.g. the context can be the communication structure or the functional units that are used. *Hyperreconfiguration steps* allow to define the potential for reconfiguration that is available to the following ordinary reconfiguration steps. Thus, hyperreconfiguration steps define the set of contexts that are available to the following (ordinary) reconfiguration steps. Such a set of available contexts is called a *hypercontext*. An (ordinary) reconfiguration takes place always within the actual hypercontext. A central aspect of hyperreconfigurable architectures is that the costs (i.e. the time or the amount of bits necessary to be loaded onto the architecture) for a reconfiguration step depend on the actual hypercontext.

An algorithm/computation is characterized by a sequence of context requirements that specify which reconfigurable features are needed during runtime for every reconfiguration step on a hyperreconfigurable machine. The context requirements are minimal requirements that have to be satisfied in order to guaranty a successful computation. Examples are the number of switches that are available for reconfiguration in order to satisfy the routing demands or the number of functional units that are available for reconfiguration to satisfy the computation demands. Note, that the actual demand of a computation during runtime might depend on the data and cannot be determined exactly in advance. In that case the context requirements are worst case upper bounds. When the meaning is clear we call the context requirements of an algorithm/computation sometimes simply its contexts. The reason is that each context requirement corresponds to exactly one new context that will be realized during runtime by a reconfiguration operation. Reconfiguration into a new context can only be realized at runtime when the machine is in a state (more exactly in a hypercontext) that provides the necessary reconfigurable features, i.e., it satisfies the corresponding context requirement.

Let $\mathcal{C}$ be the set of possible context requirements for a reconfigurable machine. An algorithm/computation is characterized by a sequence $C = c_1 \dots c_n$ of context requirements $c_i \in \mathcal{C}$, $i \in [1 : n]$. A *hypercontext* defines the reconfigurable features of the reconfigurable machine which are available in the current state and is characterized by the subset of $\mathcal{C}$ context requirements that can be satisfied in this state. Let $\mathcal{H}$ be the set of possible hypercontexts. For a hypercontext $h \in \mathcal{H}$ let $h(\mathcal{C}) \subset \mathcal{C}$ be the subset of context requirements that are satisfied by h. The set $h(\mathcal{C})$ is called the *context set* of h. For a sequence $c_1 \dots c_k$ of context requirements and a hypercontext h let $c_1 \dots c_k \subset h(\mathcal{C})$ denote the fact that for each context c_i, $i \in [1 : k]$, $c_i \in h(\mathcal{C})$ holds. To bring the machine into a new hypercontext a *hyperreconfiguration step* is necessary. For each hypercontext $h \in \mathcal{H}$ there exist two costs: i) $init(h)$ are the costs to perform a hyperreconfiguration that brings the machine into hypercontext h, ii) $cost(h)$ are the costs for an ordinary reconfiguration step when the machine is in hypercontext h.

Thus during the run of an algorithm/computation a machines performs operations $h_1 S_1 \dots h_r S_r$ where $h_1, \dots, h_r$ are hyperreconfigurations and S_i stands for a sequence of reconfigurations which use only those parts of the machine that are available within h_i. The following three cost models for hyperreconfigurable machines have been discussed for single task applications in [9]:

General model: The total reconfiguration time of a computation is measured as

$$\sum_{i=1}^{r}(init(h_i) + cost(h_i) \cdot |S_i|)$$

where $|S_i|$ is the length of S_i, i.e., the number of context requirements in S_i.

The next model is intended for coarse grained reconfigurable machines where the number of possible hypercontexts is not too large. It is assumed that the different hypercontexts can be ordered with respect to their computational power by a precedence relation. The precedence relation is given as a directed acyclic graph (DAG). It is assumed that there always exists a hypercontext h which satisfies every possible context requirement, i.e. $h(\mathcal{C}) = \mathcal{C}$.

DAG model: Given a DAG $G = (V, E)$ with $V = \mathcal{H}$ for a set $\mathcal{H}$ of hypercontexts and for each $h \in \mathcal{H}$ a set $h(\mathcal{C})$ such that for each edge in $(h_1, h_2) \in E$ the relation $h_1(\mathcal{C}) \subset h_2(\mathcal{C})$ holds. In addition there are costs $cost(h) > 0$ and $init(h) = w$ defined for each $h \in \mathcal{H}$ and a constant $k \geq 0$ such that for each edge $(h_1, h_2) \in E$ $cost(h_1) \leq cost(h_2)$. The total reconfiguration time of a computation is measured as

$$r \cdot w + \sum_{i=1}^{r} cost(h_i) \cdot |S_i|$$

For each context requirement $c \in \mathcal{C}$ let $c(\mathcal{H})$ be the set of minimal (with respect to the precedence relation defined by E) hypercontexts h in the DAG which satisfy $c \in h(\mathcal{C})$. be used also for fine grained reconfigurable machines. It is assumed that there exists a set of small (similar) reconfigurable units and every subset of these units can be used to define the reconfigurable machine that is available during a hypercontext. For an example each reconfigurable unit might be a switch and the set of switches defines the available part of the reconfigurable machine. The larger the routing requirements of an algorithm for a certain context, the more switches should be made available by the corresponding hypercontext at runtime. During a reconfiguration operation the state of each available switch has to defined. Hence, the costs of reconfiguration are simply the number of available reconfigurable units.

Switch model: Given a set of reconfigurable units or switches $X = \{x_1, \ldots, x_n\}$, the set of context requirements $\mathcal{C}$ and the set of hypercontexts $\mathcal{H}$ with $\mathcal{C} = \mathcal{H} = 2^X$, i.e., $\mathcal{C}$ and $\mathcal{H}$ are the set of all subsets of X, and a sequence $C = c_1 \ldots c_m$ of context requirements with $c_i \in \mathcal{C}$, $i \in [1 : m]$. For switch $x \in X$ the relation $x \in h(\mathcal{C})$ holds, when $x \subset h$. Let $cost(h) = |h|$, where $|h|$ is the size of h, i.e. the number of switches available in h and $init(h) = w > 0$ for each $h \in \mathcal{H}$. The total reconfiguration time of a computation is measured as

$$r \cdot w + \sum_{i=1}^{r} |h_i| \cdot |S_i|$$

Note that in the PHC-DAG model the number of hypercontexts, i.e., the number of nodes of the graph, is part of the size $n+m$ of the problem instance. For the PHC-Switch model there exist 2^n hypercontexts but this number of not part of the size of the problem instance which is $n + m$.

3 Multi Task Hyperreconfigurable Machines

In this section we introduce models for multi task hyperreconfigurable machines. A new feature of these machines is that they can perform so called partial hyperreconfiguration operations which have an effect only for a subset of the tasks. As an example consider a machine where a hypercontext is characterized by different types of properties: i) qualitative properties (e.g., good, medium, or low routability) ii) quantitative properties (e.g., number of switches in the switch boxes) and iii) limited resources (number of I/O pins). Routability might be characterized by local wires and corresponding switches that are used only within a single task and by global wires and corresponding switches that can be used by all tasks. So, there can be different types of reconfigurable resources that are available on a hyperreconfigurable machine and they are used differently by the tasks. In the following we identify several aspects that are important for the design of multi task hyperreconfigurable architectures. For a multi task environment where several tasks run in parallel three types of hyperreconfigurable resources can be distinguished.

- *Private global resources*: Reconfigurable resources that have to be shared between the tasks, i.e. each task uses some amount of the resource (which has to be assigned to it) and where the total amount that is available and the assignment to the tasks is defined by the hypercontext. For example I/O units might be a resource that has to be shared between the tasks. Thus the number of I/O units available in the hypercontext has to be at least as large as the maximum total number of I/O units that is used by all tasks during any context under this hypercontext.

- *Public global resources*: Reconfigurable resources that can be used by all tasks at the same time and according to the same quality as defined by the actual hypercontext. As an example assume that the type of switches that is actually available on an FPGA is defined by the hypercontext. Then, when tasks run in disjoint areas of the FPGA they all can do their internal routing independently from the other tasks but all have the same type of switches available within their own area.

- *Local resources*: Reconfigurable resources for which the quality or amount that is available for a task can be defined during hyperreconfiguration independently from the quality or amount that is used by the other tasks. An example is an FPGA where tasks run in disjoint areas and where the type of switches available in the area of a task can be defined independently from the type of switches available in areas of other tasks.

Note, that private global and local resources differ in whether they can change their ownership. We assume here that local resources are assigned fixed to a task at initialization. In contrast, the private global resources are assigned to the tasks by global hyperreconfigurations. To define partially hyperreconfigurable machines we distinguish between two types of hyperreconfigurations:

- *Global hyperreconfigurations* determine all available private and public) global hyperreconfigurable resources.
- *Local hyperreconfigurations* determine all available local hyperreconfigurable resources and also which of the public global resources that are assigned to a task (by a global hyperreconfiguration) are available.

Note, that private global resources are involved in the global and the local hyperreconfigurations. The global hyperreconfiguration defines the maximal availability (for the local hyperreconfigurations) and the assignment to the tasks. For example it can be defined that altogether 12 I/O-units are available from which 5 are assigned to task 1. The local hyperreconfiguration can determine to which extend the assigned private global resources are available for reconfiguration. In the example in a local hyperreconfiguration it can be defined that only 3 (of the at most 5) I/O-units for task 1 are actually available and can reconfigured.

Depending on the extend to which a hyperreconfigurable machine allows partial (hyper)reconfigurations we distinguish three types of partially hyperreconfigurable machines:

- A hyperreconfigurable machine is called *partially reconfigurable* when a subset of the tasks can perform a reconfiguration without interruption of the computation of the other tasks but hyperreconfigurations can only be done for all tasks at a time.
- A hyperreconfigurable machine is called *partially hyperreconfigurable* when a subset of the tasks can perform local hyperreconfigurations and reconfigurations without interrupting the computations of the other tasks.
- A hyperreconfigurable machine is called *restricted partially hyperreconfigurable* when a subset of the tasks can perform local hyperreconfigurations without interruption of the computations of the other tasks but reconfigurations can only be done for all tasks at a time.

For partially hyperreconfigurable machines two types of hypercontexts can be distinguished:

- *Global hypercontexts* define the available global hyperreconfigurable resources and the assignment of private global resources to the tasks.
- *Local hypercontexts* define the available local hyperreconfigurable resources for the different tasks. The *extended local hypercontexts* define the local and the private global resources (within the limits of what has been assigned to a task) that are available for the different tasks.

It is assumed in this paper that during a global hyperreconfiguration step no task can perform computations and the old extended local hypercontext and the context are no longer valid afterwards. The new extended local hypercontext and a new context have to be defined after a global hyperreconfiguration. Thus a global hyperreconfiguration has a synchronizing effect. For reconfigurations and partial hyperreconfigurations different modes of synchronization are possible. We distinguish the following three modes of synchronization between the tasks for (partially) hyperreconfigurable machines:

- *Hypercontext synchronized*: Partial hyperreconfigurations are synchronized between all tasks so that no task can perform computations during a partial hyperreconfiguration no matter whether the task actually executes a partial hyperreconfiguration or is idle.
- *Context synchronized*: Reconfigurations are synchronized between all tasks so that no task can perform computations during reconfigurations no matter whether the task actually executes a reconfiguration or is idle.
- *Fully synchronized*: The machine is hypercontext synchronized and context-synchronized.
- *Non-synchronized*: The machine is neither hypercontext synchronized nor context-synchronized.

In this paper we assume that synchronization always means explicit synchronization in the form of a barrier synchronization. Hence a program for a task contains statements for (partial) hyperreconfigurations and for reconfigurations and the tasks wait until all tasks arrive at their corresponding next (hyper)reconfiguration statement. In a synchronized machine a no-hyperreconfiguration (or no-reconfiguration) statement is used by those tasks that do not perform a corresponding (hyper)reconfiguration.

Note that public global resources exist in the partially (hyper)reconfigurable models only when they are context synchronized or fully synchronized. The reason is that a reconfiguration of the public global resources (potentially) influences all tasks and therefore has to be executed synchronously.

4 Cost models

In this subsection we extent the three models for measuring the runtime of an algorithm/computation as described in Section 2 to partially hyperreconfigurable multi task machines. Since the reconfiguration times of some tasks can overlap with the computation times of other tasks it is not enough to consider only the reconfiguration times on multi task machines.

Another important aspect that is new in the multi task environment for the definition of suitable cost function is whether the reconfiguration bits can be loaded onto the machine in parallel for all tasks or not. We distinguish whether a reconfiguration operation is done

- *task parallel*, i.e., the reconfiguration bits for the tasks (i.e. for the private global and local resources) and the reconfiguration bits for the public global resources can be uploaded in parallel onto the machine, or
- *task sequentially*, i.e., all reconfiguration bits for the tasks (and for the public global resources) have to be uploaded sequentially onto the machine.

Analogous definitions can be made for partial hyperreconfiguration operations and for the reconfiguration bits that correspond to private global resources of global hyperreconfiguration operations. For the cost models that are described in the following we assume that non-synchronized operations are always executed task parallel. For synchronized operations either way is possible and we describe cost functions for both cases. Due to the limited space and for ease of description we assume in most cases that the costs for global hyperreconfiguration operations are constant.

Let H^{pub} be the set of public global resources, H^{priv} be the set of private global resources, and H^{loc} be the set of local resources. Further, let $\mathcal{H}$ be the set of global contexts, $\mathcal{H}^{loc}$ the set of local hypercontexts and $\mathcal{H}^{loc,priv}$ is the set of extended local hypercontexts. Each global hypercontext $h \in \mathcal{H}$ is a vector $(h_0, h_1, \ldots, h_m) \in (H^{pub}, (H^{priv})^m)$ where h_0 defines the available public global resources (if this type of resources exist, otherwise h has the form $(h_1, \ldots, h_m) \in (H^{priv})^m$) and h_j defines the assignment of private global resources to task T_j, $j \in [1 : m]$. Similarly, each extended local hypercontext $h^{loc,priv} \in \mathcal{H}^{loc,priv}$ is a vector $((h_1^{loc}, h_1^{priv}), \ldots, (h_m^{loc}, h_m^{priv})) \in (H^{loc} \times H^{priv})^m$ where h_j^{loc} defines the available local resources and h_j^{priv} the available private global resources that are owned by task T_j, $j \in [1 : m]$ (and therefore can be changed in a local hyperreconfiguration by task T_j). For a given hypercontext $(h_0, h_1, \ldots, h_m)$ and a given (fixed) assignment $(f_1^{loc}, \ldots, f_m^{loc}) \in (\mathcal{H}^{loc})^m$ of local resources to the tasks an extended local context $((h_1^{loc}, h_1^{priv}), \ldots, (h_m^{loc}, h_m^{priv}))$ is *valid* only if $h_j^{priv} \subset h_j$ and $h_j^{loc} \subset f_j^{loc}$ for all $j \in [1 : m]$. For a global hypercontext $h \in \mathcal{H}$ let $\mathcal{H}_h^{loc,priv}$ be the set of extended local hypercontexts that are valid under h. How the costs for performing a hyperreconfiguration are counted depends on whether this is done task parallel or task sequentially and synchronized or not. Examples are given in the following. Assume that m tasks $T_1, \ldots, T_m$ run on the machine, $(f_1^{loc}, \ldots, f_m^{loc}) \in (\mathcal{H}^{loc})^m$ is the assignment of local resources to tasks and T_j, $j \in [1 : m]$ executes between global hyperreconfiguration $h = (h_0, \ldots, h_m)$ and the next global hyperreconfiguration h' a sequence $(h_{j,1}^{loc}, h_{j,1}^{priv})S_{j,1} \ldots (h_{j,n_j}^{loc}, h_{j,n_j}^{loc})S_{j,n_j}$, $n_j \geq 1$ of valid local hyperreconfiguration and reconfiguration operations where $(h_{j,i}^{loc}, h_{j,i}^{priv}) \in \mathcal{H}^{loc,priv}$ is a local hyperreconfiguration and $S_{j,i}$ is a sequence of context requirements, $i \in [1 : n_j]$.

4.1 The Asynchronous Case

Here we describe cost models for non-synchronized partial hyperreconfigurable machines (but recall that global hyperreconfigurations are always barrier synchronized) where reconfigurations and partial hyperreconfigurations are executed task parallel.

1. General Multi Task model: The maximal total time for (hyper)reconfigurations of all tasks from h to h' is

$$init(h) + \max_{j=1}^{m}\{\sum_{i=1}^{n_j}(init(h_j, f_j^{loc}) + cost(h_{i,j}^{loc}, h_{i,j}^{priv}) \cdot |S_{j,i}|)\}$$

where $init(h)$ are the cost for a global hyperreconfiguration and $init(h_j, f_j^{loc})$ is the cost for a local hyperreconfiguration for task T_j and $cost(h_{i,j}^{loc}, h_{i,j}^{priv})$ are the reconfiguration costs for task T_j. Note, that we assume that after a global hyperreconfiguration each task has to perform a local hyperreconfiguration.

2. Multi Task DAG (MT-DAG) model: Given are a DAG $G^{priv} = (V^{priv}, E^{priv})$ with $V^{priv} \subset 2^{H^{priv}}$ for describing the private global hypercontexts and a DAG $G_j^{loc} = (V_j^{loc}, E_j^{loc})$ with $V^{loc} \subset 2^{H^{loc}}$ for describing the local hypercontexts for the tasks (analogously as in the single context model). The costs for performing a reconfiguration for a task are such that for each edge $(g^{priv}, h^{priv}) \in E^{priv}$ and for each $h^{loc} \in V^{loc}$ the inequality $cost(h^{loc}, g^{priv}) \leq cost(h^{loc}, h^{priv})$. Similarly, for each $h^{priv} \in E^{priv}$ and for each edge $(g^{loc}, h^{loc} \in V^{loc}$ the inequality $cost(g^{loc}, h^{priv}) \leq cost(h^{loc}, h^{priv})$. Further, $init(h) = w > 0$ for $h \in \mathcal{H}$ and $init(h_j, f_j^{loc}) =$

$v_j > 0$ for $h_j \in \mathcal{H}^{priv}$, $f_j^{loc} \in \mathcal{H}^{loc}$. The maximal total time for (hyper)reconfigurations of all tasks from h to h' is

$$w + \max_{j=1}^{m}\{\sum_{i=1}^{n_j}(v_j + cost(h_{1,j}^{loc}, h_{i,j}^{priv}) \cdot |S_{j,i}|)\}$$

3. Multi Task Switch (MT-Switch) model: Given is a set of private global reconfigurable units (or switches) $X^{priv} = \{x_1, \ldots, x_u\}$ and a set of local reconfigurable units (switches) $X^{loc} = \{z_1, \ldots, z_v\}$. Then $H^{priv} = X^{priv}$ and $H^{loc} = X^{loc}$, $(f_1^{loc}, \ldots, f_m^{loc})$ is a partition of a subset of X^{loc} and a global context h is a partition of a subset of X^{priv}. Cost function $cost(h^{loc}, h^{priv}) = |h^{loc}| + |h^{priv}|$ where $h^{loc} \subset X^{loc}$ and $h^{priv} \subset X^{priv}$. Further, $init(h) = w > 0$ for $h \in \mathcal{H}$ (e.g. $w = |X| + |X^{priv}|$) and $init(h_j, f_j^{loc}) = v_j > 0$ (e.g. $v_j = |h_j| + |f_j^{loc}|$) for $h_j \in \mathcal{H}^{priv}$, $f_j^{loc} \in \mathcal{H}^{loc}$. The maximal total (hyper)reconfiguration time of all tasks from h to h' is

$$w + \max_{j=1}^{m}\{\sum_{i=1}^{n_j}(v_j + (|h_{i,j}^{loc}| + |h_{j,i}^{priv}|) \cdot |S_{j,i}|)\}$$

and a typical special case is

$$|X| + |X^{priv}| + \max_{j=1}^{m}\{\sum_{i=1}^{n_j}(|h_j| + |f_j^{loc}| + (|h_{i,j}^{loc}| + |h_{i,j}^{priv}|) \cdot |S_{j,i}|)\}$$

A model variant assumes that a hyperreconfiguration has fixed costs w plus costs for the difference between the new hypercontext h an the predecessor hypercontext h' (see [9]). The latter costs are called changeover costs. The difference is measured as the size of the symmetric difference between the sets of switches h and h' denoted by $|h \triangle h'|$. The motivation for the introduction of changeover costs are applications where only the difference information to the predecessor hypercontext has to be loaded onto the machine.

4.2 The Synchronous Case

As an example for a cost model of a synchronized machine we describe the cost model for the fully synchronized MT-Switch machine. For ease of description we assume formally that there is a partial hyperreconfiguration immediately before every reconfiguration (since a barrier synchronization model is assumed each tasks executes a local hyperreconfiguration or no-hyperreconfiguration operation). We assume further that the computation time after a reconfiguration up to the next hyperreconfiguration is the same for all tasks (if this is not the case we count the maximum computation time of the tasks between corresponding reconfigurations). An example for such a machine is a reconfigurable mesh where a reconfiguration is done at the start of each computational cycle.

Assume that each tasks performs n local (no-)hyperreconfiguration operations and n reconfigurations between global hyperreconfigurations h and h'. For $j \in [1 : m]$ and $l \in [1 : n]$ let $I_{j,l} = 0$ when the lth local (no-)hyperreconfiguration operation of task T_j is a no-hyperreconfiguration operation and otherwise let $I_{j,l} = 1$. For task T_j and $l \in [1 : n]$ let $f_j(l)$ be the number of the last local hyperreconfiguration operation in the sequence of its first l local (no-)hyperreconfiguration operations. The maximal total (hyper)reconfiguration time of all tasks when partial hyperreconfiguration and reconfiguration are executed task parallel is

$$w + \sum_{i=1}^{n}(\max_{j=1}^{m}\{I_{j,i} \cdot v_j\} + \max\{|h^{pub}|, \\ \max_{j=1}^{m}\{|h_{f_j(l),j}^{loc}| + |h_{f_j(l),j}^{priv}|\}\})$$

When partial hyperreconfiguration or reconfiguration operations are executed task sequentially basically the corresponding "$\max$" in the formula is exchanged by a "$\sum$", e.g. when partial hyperreconfiguration is done task sequentially and reconfiguration is done task parallel the costs are

$$w + \sum_{i=1}^{n}(\sum_{j=1}^{m}\{I_{j,i} \cdot v_j\} + \max\{|h^{pub}|, \\ \max_{j=1}^{m}\{|h_{f_j(l),j}^{loc}| + |h_{f_j(l),j}^{priv}|\}\})$$

5 The Synchronized MT-Switch Problem

Under the general cost model the problem to find the best time steps when to perform global and local hyperreconfigurations in order to minimize the runtime of a computation is NP-hard even for the single task environment ([9]). We show that the problem can be solved efficiently under the fully synchronized switch cost model for the multi task environment where hyperreconfigurations and reconfigurations are done task parallel. Here we show that the problem can be solved efficiently under the fully synchronized switch cost model for the multi task environment where hyperreconfigurations and reconfigurations are done task parallel. Due to space limitations we describe only the case that there are only local resources and no global resources and omit the corresponding algorithm. Recall, there are no global hyperreconfiguration in this case.

The *MT-Switch problem* is defined as follows: Given tasks T_j, $j \in [1 : m]$ and for each task a sequence of context requirements $c_{j,1} \ldots c_{j,n}$ the problem is to find the best time steps when to perform local hyperreconfigurations and to define the corresponding hypercontexts so that the total (hyper)reconfiguration time is minimal.

With a dynamic programming algorithm we can obtain the following theorem (It should be noted that the runtime can be further improved with pointer techniques as described in the full paper).

Theorem 1. *The MT-Switch problem for fully synchronized multi task hyperreconfigurable machines where partial hyperreconfiguration and reconfiguration are done task parallel can be solved in time $O(mn^4 l_m^2)$ when there are no*

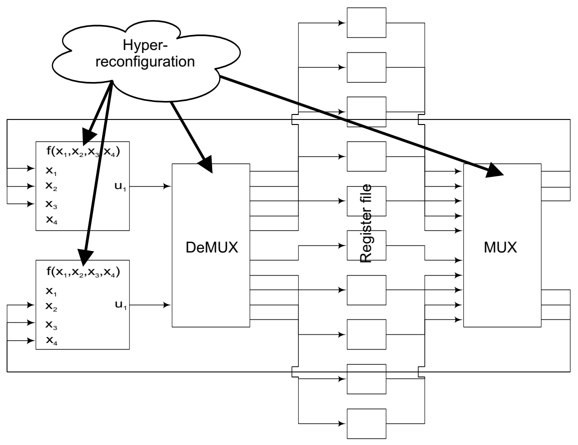


Figure 1. SHyRA architecture

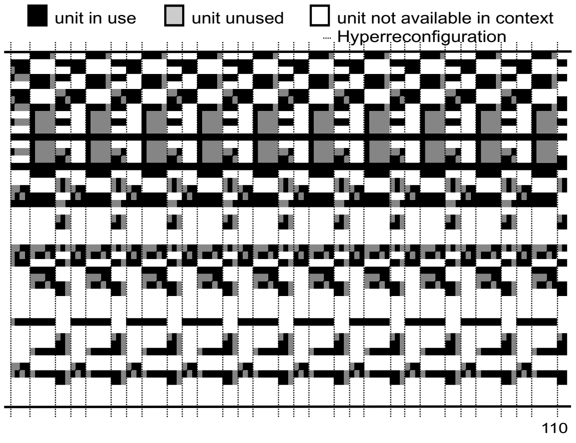
private global resources. With private global resources the problem can be solved in time $O(mn^7(l_m+g)^2)$ where g is the number of private global switches.

6 Example Architecture and Experimental Results

We define the Simple HYperReconfigurable Architecture (SHyRA) as an example of a minimalistic model of a rapidly reconfiguring machine. As depicted in Figure 1 it features 2 reconfigurable Look-Up Tables each with 3 inputs and one output. For storing signals a file of 10 registers is used which are reconfigurably connected to both LUTs by a 10:6 multiplexer and a 2:10 demultiplexer. The small number of LUTs poses a bottle neck for the test applications we have run on SHyRA and forces them to make extensive use of reconfigurations. The test applications therefore naturally lend themselves to profit from the use partial hyperreconfigurations. For the test runs SHyRA worked in fully synchronized mode and partial hyperreconfigurations were performed in task parallel mode. We compare the multiple task case where each of the four components (LUT1, LUT2, MUX, DeMUX) forms a task ($m = 4$) and partial hyperreconfiguration is possible with the single task case where all components are combined into one single task ($m = 1$).

As a first example application, a 4 bit counter with a variable upper bound was mapped onto SHyRA. The counter increments its value that is stored in the first four registers until it has reached the value stored in registers five to eight. As all operations can only be performed through the use of the LUTs it is impossible to implement the counter in one clock cycle. The design is thus time partitioned.

The resulting code was executed on a simulated SHyRA system where hyperreconfiguration was disabled. The initial counter value was 0000 and the upper bound was set to 1010. During execution each reconfiguration step was traced yielding a total of $n = 110$ reconfigurations. We analyzed the sequence of reconfigurations (seen as a sequence of $n = 110$ reconfiguration requirements) under the MT-Switch cost model for the multiple task and the single task case. For the MT-Switch cost model we assume that there are 48 switches (each corresponding to one of the 48 reconfiguration bits) that are local resources. For the multiple task model the tasks and corresponding number of local switches are: $T_1 = LUT1$ with $l_1 = 8$, $T_2 = LUT2$ with $l_2 = 8$, $T_3 = DeMUX$ with $l_3 = 8$, and $T_4 = MUX$ with $l_4 = 24$. (Hyper)reconfiguration costs with partial hyperreconfigurations for the multiple task case were computed using a genetic algorithm. For the single task case optimal (hyper)reconfiguration costs were computed (cmp. [9]).



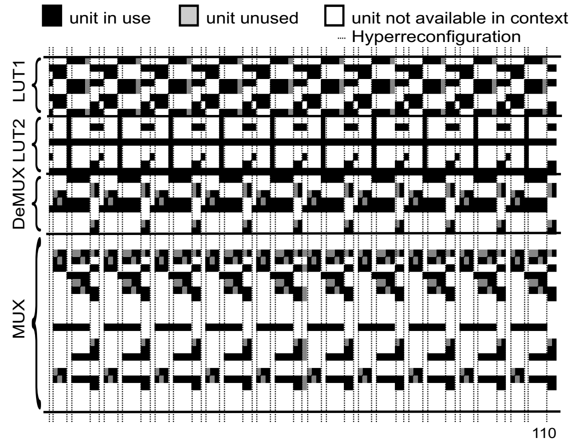


Figure 2. Hypercontexts for 4 bit counter and time of partial hyperreconfigurations for the single task case (upper) and the multiple task case (lower)

Figure 2 depicts the resulting sequences of contexts and the time steps when hyperreconfigurations were done. 50 partial hyperreconfiguration steps were used for the multiple tasks case and 30 hyperreconfiguration steps for the

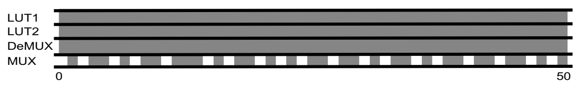


Figure 3. 50 partial hyperreconfiguration operations for 4 bit counter for the multiple tasks case (partial hyperreconfiguration (black), no-hyperreconfiguration (white))

single task case. The total reconfiguration costs when hyperreconfiguration is disabled were 5280. This has to be compared with total (hyper)reconfiguration costs of 3761 for the single task case and costs of 2813 for the multiple task case (this is 71.2% respectively 53.3% of the reconfiguration costs without using hyperreconfiguration).

Figure 3 shows which tasks have performed a hyperreconfiguration and which performed a no-hyperreconfiguration operation. Note, that since tasks T_1, T_2, and T_3 have the same size of local context (i.e., $l_1 = l_2 = l_3$) and hyperreconfigurations are done task parallel either all four tasks perform a partial hyperreconfiguration or tasks $T_1, \dots, T_3$ perform a partial hyperreconfiguration. The results show that partial hyperreconfiguration is a promising concept that has a potential to increase the speed of runtime reconfiguration.

7 Conclusion

The concept of hyperreconfigurable architectures that can adapt their ability of reconfiguration during runtime in order to speed up ordinary reconfiguration steps has been extended in this paper to multi task environments. Several models for partially and dynamically hyperreconfigurable architectures have been proposed. It was shown that the problem to find optimal (hyper)reconfigurations can be solved by a polynomial time dynamic programming algorithm for the switch cost model when (hyper)reconfigurations are done fully synchronized. The introduced concepts for partial hyperreconfiguration were illustrated by a simulated example architecture that has different reconfigurable units (2 LUTs, multiplexer, and demultiplexer). Experimental results for a small application (4-bit counter) that was run on the example architecture under the multiple tasks switch cost model show how partial hyperreconfiguration can lead to reduced (hyper)reconfiguration costs.

References

[1] K. Bondalapati, V.K. Prasanna: Reconfigurable Computing: Architectures, Models and Algorithms. In Proc. Reconfigurable Architectures Workshop, IPPS, (1997).

[2] K. Compton, S. Hauck: Configurable Computing: A Survey of Systems and Software. *ACM Computing Surveys*, 34(2):171–210 (2002).

[3] A. Dandalis, V. K. Prasanna: Configuration Compression for FPGA-based Embedded Systems. In Proc. ACM International Symposium on Field-Programmable Gate Arrays, 173–182, (2001).

[4] A. Ejnioui, N. Ranganathan: Routing on Switch Matrix Multi-FPGA-Systems. In: 13th International Conference on VLSI Design, 248–253 (2000).

[5] C. Haubelt, J. Teich, K. Richter, and R. Ernst: System Design for Flexibility. In Proc. 2002 Design, Automation and Test in Europe, 854–861 (2002).

[6] S. Hauck, Z. Li, and J.D.P. Rolim: Configuration Compression for the Xilinx XC6200 FPGA. *IEEE Trans. on Computer-Aided Design*, 8:1107–1113 (1999).

[7] P. Kannan, S. Balachandran, and D. Bhatia: On Metrics for Comparing Routability Estimation Methods for FPGAs. Proc. 39th Design Autom. Conf., 70–75 (2002).

[8] M. Koester, J. Teich: (Self-)reconfigurable Finite State Machines: Theory and Implementation. In Proc. Design. Autom. and Test in Europe, 559–566 (2002).

[9] S. Lange, M. Middendorf: Hyperreconfigurable Architectures and the Partition into Hypercontexts Problem. Submitted, (2003).

[10] K.K. Lee, D.F. Wong: Incremental Reconfiguration of Multi-FPGA Systems. In Proc. Tenth ACM Int. Symp. on Field Programmable Gate Arrays, 206–213 (2002).

[11] T.-M. Lee, J. Henkel, and W. Wolf: Dynamic Runtime Re-Scheduling Allowing Multiple Implementations of a Task for Platform-Based Designs. In Proc. 2002 Design, Automation and Test in Europe, 296–301 (2002).

[12] M. Middendorf: Models and Architectures for Hyperreconfigurable Hardware. 1st Meeting DFG Priority Programme 1148, Stuttgart, 2003, manuscript.

[13] R.P.S. Sidhu, S. Wadhwa, A. Mei, and V.K. Prasanna: A Self-Reconfigurable Gate Array Architecture. Proc. FPL, Springer,LNCS 1896, 106–120 (2000).

[14] A. Singh, G. Parthasarathy, and M. Marek-Sadowska: Interconnect Resource-Aware Placement for Hierarchical FPGAs. ICCAD 2002, 132–136 (2001).

[15] M. Teich, S. Fekete, and J. Schepers: Compile-Time Optimization of Dynamic Hardware Reconfigurations. Proc. Int. Conf. on Par. and Distr. Proc. Techniques and Appl. (PDPTA'99), Las Vegas, U.S.A., (1999).

[16] S. Wadhwa, A. Dandalis: Efficient Self-Reconfigurable Implementations Using On-chip Memory. Proc. FPL, LNCS 1896, 443–448 (2000).