Hyperreconfigurable Architectures for Fast Run Time Reconfiguration[*]

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

Dynamically reconfigurable architectures or systems are able to reconfigure their function and/or structure to suit changing needs of a computation during run time. The increasing flexibility of modern dynamically reconfigurable systems improves their adaptability but also makes fast reconfiguration difficult because of the large amount of necessary reconfiguration information. However, even when a computation uses this flexibility it will not use it all the time. Therefore, we propose to make the potential for reconfiguration itself reconfigurable. This allows for speeding up reconfiguration operations during phases where only parts of the total flexibility are required. Such architectures are called hyperreconfigurable and use two types of reconfiguration operations: hyperreconfigurations for changing the reconfiguration potential and ordinary reconfigurations for actually configuring a new context for a computation.

1 Hyperreconfigurable Architectures

Dynamically reconfigurable architectures can adapt their function and/or structure to suit the changing needs of a computation during run time (e.g., [1,2]). A problem is the tradeoff between flexibility and the amount of information needed for reconfiguration to define the new state of the system. Moreover, the increasingly higher integration of reconfigurable hardware requires increased bandwidths for transferring the reconfiguration information. Modern FPGAs for example need several megabytes of reconfiguration data for a single reconfiguration step. This large amount of data transfer makes run time reconfigurations time critical operations, especially, for computations which exploit the full capacity of these architectures by frequent reconfigurations. Different approaches have been proposed in the literature to cope with this problem, e.g., compression methods for the stream of reconfiguration bits ([3,4]) and self-reconfigurability (see [5,6,7].

[*]Financial support by the German Research Foundation (DFG) through project ”Models and Algorithms for Hyperreconfigurable Architectures”

In this paper we propose a new approach to make run time reconfiguration faster by defining a new type of reconfigurable architectures. We use the fact that algorithms or computations typically consist of different phases where during each phase only a fraction of the reconfiguration potential of the underlying architecture is needed. The idea is to make the reconfiguration potential itself reconfigurable. The smaller the actual reconfiguration potential of an architecture is the smaller will the amount of reconfiguration information be that has to be transferred during reconfiguration and the faster will a reconfiguration step be. Due to space limitations this paper focuses on the introduction of the concept of hyperreconfigurable architectures.

We call dynamically reconfigurable architectures and systems which allow to alter the reconfiguration potential during run time *hyperreconfigurable architectures*. Hyperreconfigurable architectures have two types of reconfiguration steps: (*ordinary*) *reconfiguration steps* are used to actually define a new configuration of the system and *Hyperreconfiguration steps* are used for defining the actual reconfiguration potential that is available for the ordinary reconfiguration steps. The actual state of the system that can be changed by reconfiguration is called *context*. Thus, a hyperreconfiguration step defines the set of contexts that is available for the (ordinary) reconfiguration steps. Such a set of available contexts is called a *hypercontext*. With ”available” we assign those reconfigurable resources that are activated by the hypercontext and therefore are available for reconfiguration. If a reconfiguration needs resources that are not included in the hypercontext they have to be activated/included by a hyperreconfiguration. We assume here that a reconfiguration step requires reconfiguration information for all activated resources (even when it says that the resource is not used in the context). Examples for hypercontexts are the set of activated reconfigurable units that are available for reconfiguration or the available routing resources. Often the context requirements might be an estimated upper bound on the requirements that will actually be needed during run time. Formal models for hyperreconfigurable architectures will be discussed in the next section.

2 Formal Models for Hyperreconfiguration

We introduce ideal models that allow us to consider algorithmic aspects for hyperreconfigurable architectures. These models can be made more specific to describe concrete architectures. We assume that an algorithm/computation is characterized by a sequence of *context requirements* each describing the resource requirements that it needs for a corresponding reconfiguration step. Hence the number of context requirements equals the number of reconfiguration steps. Since the actual reconfiguration steps might depend on data that are only available at run time a context requirement specifies the (estimated) maximal set of resources that could possibly be needed. When the meaning is clear we call a context requirement simply context. A reconfiguration into a new context can in general only be realized during run time when the machine is in a hypercontext that contains all contexts possible according to the corresponding context requirement, i.e., the hypercontext *satisfies* the corresponding context requirement.

In the following we describe cost models to count the (hyper)reconfiguration time. Let $\mathcal{C}$ be the set of possible context requirements for a reconfigurable machine and $C = c_1 \dots c_m$, $c_i \in \mathcal{C}$ be the sequence of context requirements that characterizes an algorithm/computation. A *hypercontext* is a state of the machine which is characterized by the subset of $\mathcal{C}$ context requirements that are satisfied when the machine is in this state. At any time exactly one hypercontext is realized on the machine. 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 requirement c_i, $i \in [1:k]$ $c_i \in h(\mathcal{C})$ holds. In order change the machine's current hypercontext a *hyperreconfiguration step* is necessary. For each hypercontext $h \in \mathcal{H}$ two cost measures are defined: i) $init(h)$ is the cost of performing a hyperreconfiguration that brings the machine into hypercontext h ii) $cost(h)$ denotes the cost of an ordinary reconfiguration step when the machine is in hypercontext h.

Then a computation is characterized by a partition of C into substrings $S_1, \dots, S_r$ (i.e. $C = S_1 \dots S_r$) and hypercontexts $h_1, \dots, h_r$, $r \geq 1$ such that $S_i \subset h_i(\mathcal{C})$ and $\sum_{i=1}^{r}(init(h_i) + cost(h_i) \cdot |S_i|)$ are the costs where $|S_i|$ is the length of S_i, i.e., the number of context requirements in S_i. During the run of the algorithm/computation that is executed the machine performs the following reconfiguration operations: $h_1 S_1 \dots h_r S_r$ where S_i stands for a sequence of $|S_i|$ reconfigurations which use only those parts of the machine which are available within the hypercontext h_i. We assume that a hyperreconfiguration is always performed before the first reconfiguration step.

We define two variants of the model — the first variant called *DAG-model* is for coarse grained reconfigurable machines where the set of possible hypercontexts is not too large and where different reconfigurable submachines (hypercontexts) can be defined that can be ordered with respect to their computational power by a precedence relation. A directed acyclic graph (DAG) where each node is a hypercontext is used to describes the precedence relation between the hypercontexts. We assume that a hypercontext h exists that satisfies all possible context requirements, i.e. $h(\mathcal{C}) = \mathcal{C}$. Formally, given a DAG $G = (V, E)$ with $V = \mathcal{H}$ 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 let $cost(h) > 0$ and $init(h) = w$ for each $h \in \mathcal{H}$ and a constant $k \geq 0$ such that for each edge $(h_1, h_2) \in E$ $cost(h_1) \leq cost(h_2)$. Then a computation is characterized by a partition of C into substrings $S_1, \dots, S_r$, $r \geq 1$ (i.e. $C = S_1 \dots S_r$) and hypercontexts $h_1, \dots, h_r$ such that $S_i \subset h_i(\mathcal{C})$ and the total (hyper)reconfiguration costs are $r \cdot w + \sum_{i=1}^{r} cost(h_i) \cdot |S_i|$.

The second variant called *Switch-model* is for fine grained machines. We assume that there exists a set of small (similar) reconfigurable units and every subset can be used to define the reconfigurable machine that is available during a hypercontext. For example each unit might be a switch in a switch box on an FPGAs and the more routing requirements an algorithm has for a context, the more switches should be available in the hypercontext for reconfiguration during run time. For reconfiguration the state of each available switch has to be defined. Thus the cost for reconfiguration is the number of available units plus overhead cost. Formally, let $X = \{x_1, \dots, x_n\}$ a set of switches and define $\mathcal{C} = \mathcal{H} = 2^X$, i.e., $\mathcal{C}$ and $\mathcal{H}$ equal the set of all subsets of X. For context $x \in X$ 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 $init(h) = n$ for $h \in \mathcal{H}$. A computation is characterized by a partition of C into substrings $S_1, \dots, S_r$, $r \geq 1$ (i.e. $C = S_1 \dots S_r$) and hypercontexts $h_1, \dots, h_r$ such that $S_i \subset h_i(\mathcal{C})$ and the total (hyper)reconfiguration costs are $r \cdot n + \sum_{i=1}^{r} |h_i| \cdot |S_i|$.

Future work: Multi task hyperreconfigurable machines and algorithmic aspects will be discussed elsewhere.

References

[1] K. Bondalapati, V.K. Prasanna. Proc. RAW, 1997
[2] K. Compton, S. Hauck, ACM Computing Surveys, 34(2): 171–210, 2002
[3] A. Dandalis, V. K. Prasanna: Proc. ACM International Symposium on FPGAs, 173–182, 2001
[4] S. Hauck, Z. Li, J.D.P. Rolim, IEEE Trans. on CAD of Integrated Circuits and Systems, 8:1107–1113, 1999
[5] M. Koester, J. Teich, Proc. DAC Europe, 559–566, 2002
[6] R.P.S. Sidhu, S. Wadhwa, A. Mei, V.K. Prasanna, Proc. FPL, 106–120, 2000
[7] S. Wadhwa, A. Dandalis, Proc. FPL, 443–448, 2000