Multi-level Reconfigurable Architectures in the Switch Model

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

In this paper we study multi-level dynamically reconfigurable architectures. These are extensions of standard reconfigurable architectures where ordinary reconfiguration operations correspond to the lowest reconfiguration level. On each higher reconfiguration level the reconfiguration capabilities of the reconfigurable resources that are available on the level directly below can be reconfigured. We show that the problem to find optimal reconfigurations with an arbitrary number of reconfiguration levels can be found in polynomial time for the switch cost model. The problem of finding the optimal number of reconfiguration levels is shown to be solvable in polynomial time on homogeneous multi-level architectures but it becomes NP-hard for heterogeneous multi-level architectures. Moreover, we present experimental results for some example problems on a simple test architecture.

1 Introduction

Reconfigurable architectures that can be reconfigured dynamically on two different levels have been proposed in [3]. These architectures allow to adapt the amount of reconfiguration possibilities that are available for (ordinary) reconfigurations to the actual needs. A small amount of reconfiguration possibilities offers less flexibility but has the advantage that dynamic reconfiguration is faster because less reconfiguration bits are necessary to define the new state of the architecture. The lower reconfiguration level of a 2-level reconfigurable architecture corresponds to ordinary reconfiguration operations. On the upper reconfiguration level the reconfiguration capabilities of the reconfigurable resources that are available for the lower level reconfiguration can be reconfigured. The upper level reconfiguration operations were called hyperreconfigurations in [3]. Accordingly there exist two types of (reconfiguration) contexts: i) the ordinary context is defined by the reconfiguration bits that have been loaded onto the architecture during the last reconfiguration operation and ii) the hypercontext is defined by the reconfiguration bits that have been loaded during the last hyperreconfiguration operation. This definition of 2-level reconfiguration is different from the two-level reconfiguration scheme of the MATRIX architecture [5]. MATRIX uses a two level configuration scheme where traditional "instructions" are seen as the lower reconfiguration level. The higher level reconfiguration is used to configure the architectural organization for a computation and the corresponding reconfiguration data are called metareconfiguration data.

Several aspects of 2-level reconfigurable architectures have been studied so far. The problem to find for an algorithm that is characterized by a sequence of ordinary reconfiguration operations the best time steps when to perform upper level reconfiguration operations and to determine the corresponding hypercontexts was studied in [3] and called Partition into Hypercontexts (PHC) problem. It was shown that PHC is NP-hard in general but for the so called switch model of 2-level reconfiguration it can be solved in polynomial time. The use of a context cache in 2-level reconfigurable architectures was studied in [2]. It was shown that the PHC problem with cache is NP-complete even in the switch model of 2-level reconfiguration no matter which cache replacement strategy is used. Heuristics to select the contexts that are stored in the upper level cache have been proposed. The fundamental design problem to find the best level of granularity that should be provided for the upper level reconfiguration operations was addressed in [4].

It has been argued in [3] that architectures with more than two reconfiguration levels might be advantageous to make dynamic reconfiguration even faster. But so far no detailed investigations have been done to

1-4244-0054-6/06/$20.00

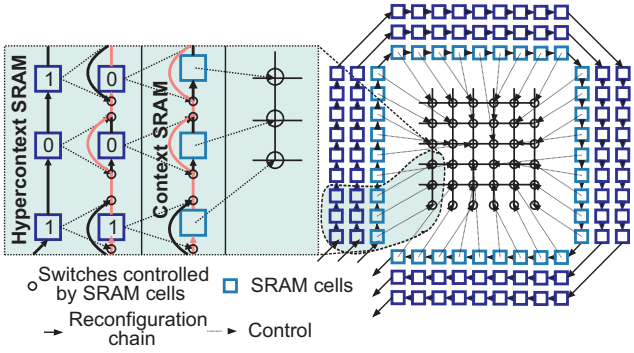

Figure 1. 3-level reconfigurable switchbox

explore the concept of multi-level reconfiguration for more than two levels of reconfiguration.

In this paper we introduce a formal model for multi-level reconfigurable architectures and study algorithmic problems for their use. In particular, we define a multi-level reconfigurable Switch model architecture and investigate versions of the PHC problem for this architectures. We extend the results of [3] and show that the PHC problem can be solved optimally in polynomial time also for Switch model reconfigurable architectures with more than 2 levels.

2. Multi-Level Reconfigurable Architectures

In this section we introduce multi-level reconfigurable machines (as an extension of 2-level reconfigurable architectures as introduced in [3]). We start with intuitive definitions to introduce the concept and then present formal models. Multi-level reconfigurable architectures have two types of reconfiguration operations. The lowest (or first) level of reconfiguration corresponds to ordinary reconfiguration operations. Thus, during a lowest level reconfiguration operation (*ordinary* reconfiguration operation) the context (in the sense of [1]) of an algorithm is defined. A context determines, e.g., the connections for communication or the functions that are computed by Look-Up Tables.

Let $r \geq 2$ be the number of reconfiguration levels. The kth level of reconfiguration, $r \geq k \geq 2$ is used to define the capabilities of the reconfigurable resources or the availability of these resources for the subsequent reconfiguration steps on level $k-1$. In other words, k-level reconfiguration operations, $k \geq 2$ define the set of contexts that are available to lower reconfiguration levels. A central aspect of multi-level reconfigurable architectures is that the cost (i.e., the reconfiguration time or the amount of bits necessary to be loaded onto the architecture) of a $k-1$-level reconfiguration operation depend on the current k-level context as explained in the following, $k \geq 2$. The k-level reconfiguration operations, $k \geq 2$, are also called hyperreconfigurations and the corresponding contexts hypercontexts ([3]).

In this paper we assume that an algorithm/computation can be characterized by a sequence of context requirements that specify which reconfigurable features are needed for every ordinary reconfiguration step during run time on the multi-level reconfigurable machine. The context requirements are minimal requirements that have to be satisfied in order to guaranty a successful computation. One example is that a certain set of switches in the switchboxes is required for reconfiguration in order to satisfy the routing demands. Another example is a minimum required number of functional units that are available for reconfiguration to satisfy the computational demands. Note that the actual reconfiguration of a computation during run time might depend on the data and therefore we do not assume that it can be determined exactly in advance. Thus not all of the switches that were required for a reconfiguration step are necessarily being used during run time in the corresponding reconfiguration step. 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. Before we describe the formal model we give an example.

Example. Consider a 3-level reconfigurable switchbox with 6 inputs, 6 outputs and 36 reconfigurable switches s_{ij}, $i,j \in [1:6]$ (see Figure 1). There exists a chain of S-RAM cells — called reconfiguration chain — that defines the state of the switches. During a reconfiguration operation the reconfiguration bits are shifted into this register chain. There exist two other chains of S-RAM cells called 1st- and 2nd-hyperreconfiguration chain. These chains are used to define whether an S-RAM cell is included in the reconfiguration chain (respectively the 1st-level hyperreconfiguration chain) or is shortcut.

During a k-level reconfiguration operation the hyperreconfiguration bits are shifted into the kth register chain, $k \in [2:3]$. Assume that an algorithm A makes six 1st-level reconfiguration operations and therefore can be characterized by a sequence of six context requirements $c_1c_2c_3c_4c_5c_6$. We assume that the requirement for the first two contexts is that all 6 switches on the diagonal are needed, i.e. $c_1 = c_2 = \{s_{11}, s_{22}, s_{33}, s_{44}, s_{55}, s_{66}\}$ (Figure 2 (a)), the next two reconfigurations need only the upper third of the diagonal, i.e. $c_3 = c_4 = \{s_{11}, s_{22}\}$ (Figure 2 (b)), and for the

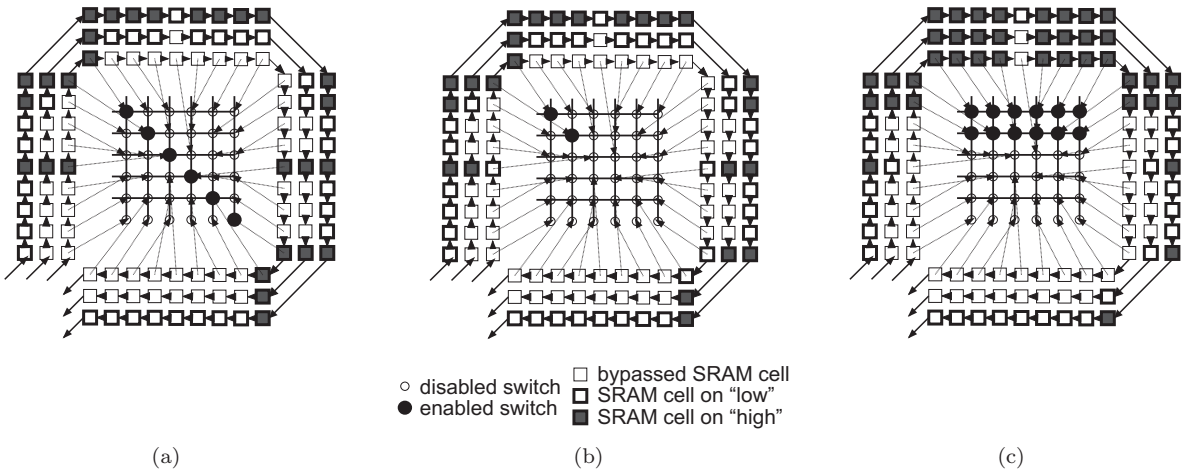

Figure 2. The example 3-level reconfigurable switchbox with corresponding (hyper-)reconfiguration data for reconfigurations 1-2 (a), 3-4 (b) and reconfigurations 5-6 (c)

last two reconfiguration steps only the upper two rows are required, i.e., $c_5 = c_6 = \{s_{11}, \ldots, s_{16}, s_{21}, \ldots, s_{26}\}$ (Figure 2 (c)). Note that it is not known in advance how the switches are used (i.e., on or off).

Assume that a 3rd-level reconfiguration is performed before the first reconfiguration step so that exactly those S-RAM cells that correspond to switches on the diagonal and the first two rows are included in the 2nd-level reconfiguration chain (i.e. the 3rd-level context is $\{s_{11}, \ldots, s_{16}, s_{21}, \ldots, s_{26}, s_{33}, s_{44}, s_{55}, s_{66}\}$). Assume further that a 2nd-level reconfiguration is performed so that exactly those S-RAM cells that correspond to switches on the diagonal are included in the 1st-level reconfiguration chain (i.e. the 2nd-level context is $\{s_{11}, s_{22}, s_{33}, s_{44}, s_{55}, s_{66}\}$). Observe, that only 16 bits have to be shifted into the 2nd level reconfiguration chain for this. For each of the first four 1st-level reconfiguration operations only 6 reconfiguration bits have to be shifted into the reconfiguration chain. Assume that another 2nd-level reconfiguration is made before the fifth reconfiguration step so that exactly the S-RAM cells of the switches on the first two rows are included into the 1st-level reconfiguration chain. Consequently, for each of the last two 1st-level reconfiguration operations only 12 reconfiguration bits have to be shifted into the reconfiguration chain. Altogether, a total number of 116 reconfiguration bits have to be shifted into the 1st-level reconfiguration chains: 36 3rd-level reconfiguration bits, 32 2nd-level reconfiguration bits and 48 1st-level reconfiguration bits. This is less than 6x36=216 reconfiguration bits that would have been necessary for an ordinary (i.e. 1-level) reconfigurable switchbox where each reconfiguration step requires 36 reconfiguration bits. Observe also that it would be possible to make an additional 2nd-level reconfiguration before the third reconfiguration so that only the switches $\{s_{11}, s_{22}\}$ are included in the reconfiguration chain. But this is more costly because it needs 16 additional 2nd-level reconfiguration bits and saves only 8 1st-level reconfiguration bits (4 for the each of the 3rd and 4th reconfiguration operation).

Formal model. The formal model that we introduce in the following is an extension of the 2-level reconfigurable switch model as introduced in [3]. It should be noted that it is possible to extent other models of 2-level reconfigurable machines from [3]. Since we want to concentrate on the algorithmic problems in this paper this is not done here.

A multi-level reconfigurable architecture (in the switch model) consists of a set $X_1 = \{x_1^1, \ldots, x_n^1\}$ of reconfigurable units or switches. For $r \geq 2$ reconfiguration levels the machine has for each $k \in [2:r]$ a set $X_k = \{x_1^k, \ldots, x_n^k\}$ of SRAM cells. A subsequence of $x_1^k, \ldots, x_n^k$ forms a chain (called k-level reconfiguration chain) that can be loaded from outside, $k \in [1:r]$. For $k = r$ always all SRAM cells $x_1^r, \ldots, x_n^r$ are included in the r-level reconfiguration chain. Which SRAM cells or switches are included into the k-level reconfiguration chain for $k < r$ is defined by the content of the SRAM cells on higher levels as described in the following. The state of an SRAM cell x_i^k can be used to define whether the corresponding SRAM cell one level below (i.e. x_i^{k-1}) is included in the $k-1$-level reconfiguration chain. More exactly, if SRAM cell x_i^k is included in the k-level reconfiguration chain then SRAM cell (respectively switch, for $k = 2$) x_i^{k-1} is included in the

$k-1$-level reconfiguration chain if and only if x_i^k contains a one, $i \in [1:n]$, $k \geq 2$. If SRAM cell x_i^k is not included in the k-level reconfiguration chain then SRAM cell (respectively switch, for $k=2$) x_i^{k-1} is also not included in the $k-1$-level reconfiguration chain, $i \in [1:n]$, $k \geq 2$.

The content of the k-level reconfiguration chain is called k-level context and denoted by h_1^k, $k \in [1:r]$. The set of (1st-level) context requirements $\mathcal{C} = 2^{X_1}$ is the set of all subsets of X_1. An algorithm that performs m 1st-level reconfiguration operations is characterized by a sequence $C = c_1 \dots c_m$ of context requirements with $c_i \in \mathcal{C}$, $i \in [1:m]$. The algorithm can run successfully only when each context requirement c_i is satisfied by the actual context during the corresponding 1st-level reconfiguration operation. Thus, h_1 must contain all switches in c_i during the corresponding reconfiguration step.

For this paper we assume that after a k-level reconfiguration the machine must make a reconfiguration on all lower levels, i.e. from level k-1 to level 1 (in this order), before a computation operation can be done. Thus during the run of an algorithm/computation a machines performs operations $H_1 S_1 \dots H_p S_p$ where $H_1, \dots, H_p$ stands for a sequence of hyperreconfigurations and S_i stands for a sequence of reconfigurations which use only those parts of the machine that are available within the 1st-level context as defined by H_i.

As a measure for the reconfiguration costs of an algorithm we compute the total number of reconfiguration bits that are loaded onto the machine during its run. Consider a sequence $H = h^k h^{k-1} \dots h^2$ of hyperreconfigurations, then the costs for performing these hyperreconfiguration operations are $cost(H) := |h^k| + |h^{k-1}| + \dots + |h^2|$ where $|h^j|$ denotes the number of SRAM cells (respectively switches) that are included in the j-level reconfiguration chain at the corresponding hyperreconfiguration operation. For a sequence $H_1 S_1 \dots H_p S_p$ of operations of an algorithm where H_i stands for a sequence of hyperreconfigurations and S_i stands for a sequence of reconfigurations the total reconfiguration cost of a computation is measured as $\sum_{i=1}^{p} cost(H_i) + \sum_{i=1}^{p} |h_i^1| \cdot |S_i|$ where $|h_i^1|$ is the number of switches that are included in the 1st-level reconfiguration chain after H_i.

3. The Partition into Hypercontexts Problem

An algorithmic question that emerges for multi-level reconfigurable machines and a given algorithm (that is characterized by a sequence of context requirements) is when are the best time steps for reconfiguration operations on the different levels in order to minimize the total reconfiguration costs. Also the corresponding hypercontexts have to be defined such that the context requirements of the algorithm are satisfied. This problem has been called the Partition into Hypercontexts (PHC) problem in [3]. For the switch model of reconfigurable architectures (which is the model used in this paper) the problem was called PHC-Switch. It has been shown in [3] that the PHC-Switch problem can be solved for 2-level reconfigurable architectures by dynamic programming in time $O(n \cdot m^2)$. For more general models of 2-level reconfigurable architectures the problem is NP-hard. Formally we define the PHC-Switch problem for multi-level architectures as follows.

PHC-Switch problem for multi-level architectures: Given a multi-level reconfigurable architecture in the switch model where $r \geq 2$ is the number of reconfiguration levels and $X = \{x_1, \dots, x_n\}$ is the set of switches. Given also a sequence of context requirements $C = c_1 \dots c_m$. Find a partition of C into substrings $S_1, \dots, S_p$, $p \geq 1$ (i.e. $C = S_1 \dots S_p$) and sequences $H_1, \dots, H_p$ of hyperreconfiguration operations with corresponding hypercontexts such that each context requirement is satisfied by the actual 1-level context and the total reconfiguration costs are minimal.

4. The Multi-level PHC-Switch Algorithm

In this section we describe an optimal polynomial time dynamic programming algorithm for PHC-Switch for multi-level reconfigurable architectures. First, we give a recursive version of the algorithm. Algorithm $Multi\_phc(level, hc\_cost, l, u)$ computes for reconfiguration level $level \geq 1$ the minimal costs for computing the sequence $c_l \dots c_u$ of C and under the assumption that a $level$-level reconfiguration operation has cost hc_cost. The algorithm computes a table M_{ij}, $i \in [l+1, \dots, u]$, $j \in [l, u]$ where M_{ij} are the minimal costs for computing the sequence $c_l \dots c_j$, $i \leq j \leq u$ under the assumption that i many $level$-level reconfiguration operations are used. For $level = 1$ $Multi\_phc$ computes only the standard reconfiguration costs.

Let $R_{k,j}$ be the number of switches that are contained in at least one of the context requirements $c_k, \dots, c_j$. The code for algorithm $Multi\_phc(level, hc\_cost, l, u)$ is given in Algorithm 1.

In order to proof that the PHC-Switch problem can be solved on multi-level reconfigurable architectures in polynomial time we consider an iterative version of Algorithm 1 in the proof of the following theorem.

Algorithm 1 Multi-Level PHC Algorithm $Multi\_phc(level, hc\_cost, l, u)$

1: **if** $level = 1$ **then**
2: **return** $R_{l,u} \times (u - l + 1)$ {plain minimal reconfiguration costs}
3: **else**
4: **define** matrix $M_{i,j}; i, j \in [l, u]$
5: **for all** $M_{l,k}$; $k \in [l, u]$ **do** {calculate first row}
6: $M_{l,k} = Multi\_phc\,(level - 1, R_{l,k}, l, k) + hc\_cost$
7: **end for**
8: **for all** $M_{k,k}$; $k \in [l + 1, u]$ **do** {calculate diagonal}
9: $M_{k,k} = M_{k-1,k-1} + Multi\_phc\,(level - 1, R_{k,k}, k, k) + hc\_cost$
10: **end for**
11: **for all** $i \in [l + 1, u]$ **do** {main step}
12: **for all** $j \in [i + 1, u]$ **do**
13: $M_{i,j} = \min_{k=i}^{j}(M_{i-1,k} + Multi\_phc\,(level - 1, R_{k+1,j}, k + 1, j) + hc\_cost)$
14: **end for**
15: **end for**
16: **end if**
17: **return** $\min_{i=l}^{u}(M_{i,u})$

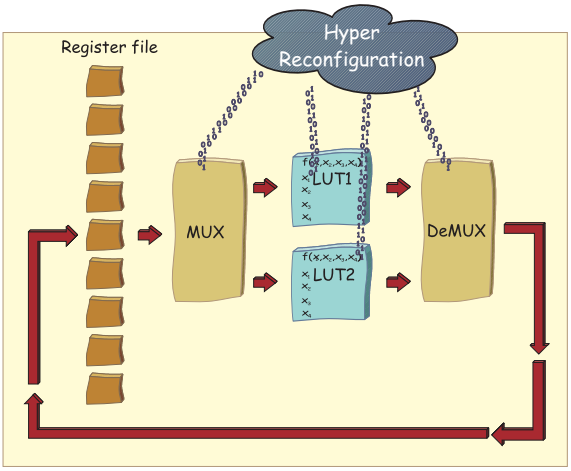

Figure 3. SHyRA

Theorem 1. *The PHC-Switch problem can be solved on multi-level reconfigurable architectures with $r \geq 2$ levels of reconfiguration optimally in polynomial time $O(nm^2 + m^2(m + n)(r - 2) + m^3)$.*

Proof. Given a multi-level reconfigurable architecture in the switch model with $r \geq 2$ reconfiguration levels, set of switches $X = \{x_1, \ldots, x_n\}$, and a sequence of context requirements $C = c_1 \ldots c_m$. The basic idea is to compute the multi-level PHC-Switch problem iteratively for an increasing number of reconfiguration levels. For this iterative version of the algorithm for each number $q \in [1 : r]$ of reconfiguration levels a table $Tab_q[rcost, l, u]$, $rcost \in [0 : n]$, $l, u \in [1 : m]$ is computed where $Tab_q[rcost, l, u]$ contains the minimum cost for computing the subsequence $c_l \ldots c_u$ on a q-level reconfigurable architecture assuming that a q-level reconfiguration level has cost $rcost$. This allows to exchange the recursive execution of $Multi\_phc$ with $q - 1$ by a lookup of the corresponding element in the Tab_{q-1}. Observe, that $Tab_q[rcost, l, u] = Tab_q[0, l, u] + rcost \times n_{Rec}$ for $n \geq rcost > 0$ where n_{Rec} denotes the number of q-level reconfigurations, which is already available from the calculation of matrix $M_{i,j}$. Hence, $Tab_q[rcost, l, u]$ for $rcost > 0$ can be computed in constant time when $Tab_q[0, l, u]$ is known. Thus on each level $q \geq 3$ the computation of Tab_q can be done in time $O(m^2(m + n))$. For the first reconfiguration level the cost of a reconfiguration is $Tab_1[rcost, l, u] = R_{l,u} \times (u - l + 1)$. It is easy to see that Tab_1 can be computed in time $O(nm^2)$. Since the cost for an r-level reconfiguration are always n it is enough to compute the entries for Tab_q with $rcost = n$. Altogether, the total computation time is $O(nm^2 + m^2(m + n)(r - 2) + m^3)$. □

A corollary to this theorem is that the best number r of reconfiguration levels for a given multi-level reconfigurable architecture in the switch model and a given sequence of context requirements can be found in polynomial time. "Best" means here to find an r such that the solution of the corresponding PHC-Switch problem is minimal. We call this problem the Reconfiguration Level Problem (RLP). We make the following observation. For a given computation a given level k of reconfiguration cannot be useful if for the solution of the corresponding PHC-Switch problem a reconfiguration on level k is always done when a reconfiguration on level $k - 1$ is done. Also there is at least one r-level reconfiguration at the beginning. Hence, it follows that for the solution of the PHC-Switch problem with the best number r of reconfiguration levels there must exist for each $k \in [2 : r - 1]$ at least one time step when a k-level reconfiguration is done but no $k + 1$-level re-

configuration is done. Hence $m+1$ is a simple upper bound for the best number of reconfiguration levels. Now we can show the following theorem.

Corollary 1. *RLP is solvable for multi-level reconfigurable architectures in polynomial time $O(nm^4 + m^5)$.*

5. Heterogeneous Multi-level

In this section we describe an extension of the multi-level reconfigurable architectures. Often there exist different types of reconfigurable resources that are used within one machine. Since the pattern of usage of the different reconfigurable resources will typically be different it cannot be expected that the same number of reconfiguration levels is optimal for all types of resources (when they are considered separately). Moreover, since each level of reconfiguration needs its own hardware resources it is proposed here to have a different number of reconfiguration levels for different reconfigurable resources. Such architectures are called here heterogeneous multi-level reconfigurable architectures. They are described further for the switch model in the following. For this model we assume that the switches (possibly) have different numbers of reconfiguration levels. All SRAM cells on the same reconfiguration levels are included into one corresponding reconfiguration chain. Thus the length of the reconfiguration chains decreases monotonically with higher levels.

Formally, a heterogeneous multi-level reconfigurable architecture (in the switch model) consists of a set $X_1 = \{x_1^1, \ldots, x_n^1\}$ of reconfigurable units or switches. Let $r \geq 2$ be the maximum number of reconfiguration levels. For each $i \in [1:n]$ there exits an $r_i \in [1:r]$ and the machine has a set $\{x_i^2, \ldots, x_i^{r_i}\}$ of SRAM cells (in the case of $r_i = 1$ the set is empty). Thus, r_i denotes the number of reconfiguration levels for switch x_i. For each $k \in [2:r]$ let X_k be the set of switches of the form x_i^k, $i \in [1:n]$. The set x_k is the set of SRAM cells that are included in the reconfiguration chain for reconfiguration level k. How the heterogeneous model works and how high are the costs for reconfiguration is defined analogously to the non-heterogeneous model (this is straightforward and can not done formally here due to limited space).

In section 4 we have shown that for a given algorithm the PHC-Switch problem can be solved on multi-level reconfigurable architectures. The main changes in the code of algorithm $Multi\_phc(level, hc\_cost, l, u)$ to obtain the corresponding algorithm $Het\_Multi\_phc(level, hc\_cost, l, u)$ for the heterogeneous case are the that the costs $R_{k,j}$ have to be defined for different levels as described in the following. Define costs $R_{level,k,j}$ as the sum over values v_i, $\in [1:n]$ where: i) $v_i = 1$ if switch x_i is used in the contexts $c_k, \ldots, c_j$ and if $level < r_i$ (recall that r_i is the number of reconfiguration levels for switch x_i), ii) $v_i = 1$ if $level = r_i$, and iii) $v_i = 0$ if $level > r_i$. It is enough to exchange each occurrence $R_{k,j}$ in the code of algorithm $Multi\_phc(level, hc\_cost, l, u)$ by $R_{level-1,k,j}$ to obtain algorithm $Het\_Multi\_phc(level, hc\_cost, l, u)$.

We can state the following result for the heterogeneous multi-level reconfigurable architectures (Observe, that the time to compute the values $R_{level-1,k,j}$ is only $O(rm^2)$).

Theorem 2. *The PHC-Switch problem can be solved on heterogeneous multi-level reconfigurable architectures with a maximum of $r \geq 2$ levels of reconfiguration in polynomial time $O(nm^2 + m^2(m+n)(r-2) + m^3)$.*

A problem that emerges with heterogeneous multi-level reconfigurable architectures is to decide for a given algorithm and each switch what is the best number of reconfiguration levels so that the solution of the corresponding PHC-Switch problem has minimum costs. We call this the Heterogeneous Reconfiguration Level Problem (H-RLP) for heterogeneous multi-level reconfigurable architectures. It was shown in Section 4 that the corresponding problem (RLP) for non-heterogeneous multi-level reconfigurable architectures is polynomial time solvable. Here we can state the following result (the proof is omitted due to limited space; the reduction is from the Maximum Clique problem).

Theorem 3. *H-RLP is NP-hard for heterogeneous multi-level reconfigurable architectures even when the maximum number of reconfiguration levels is 2.*

6. Results

In the following we evaluate the concept of multiple reconfiguration levels by applying it to four test cases. The test cases consist of reconfigurable designs implemented on the SHyRA architecture as described in [3]. The SHyRA architecture consists of a file of registers, a plane of LUTs, and a MUX and DeMUX that are used as interconnection for moving the data to the LUTs and the results back to the registers. MUX, DeMUX as well as the LUTs are multi-level reconfigurable (see Figure 3). For the tests we used an 8 bit ripple carry adder, a counter, an LED-decoder, and a simple control circuit consisting primarily of adders and counters.

The first test case is a 4 bit reconfigurable counter circuit that was implemented on a SHyRA with 2 LUTs and 8 registers. 48 reconfiguration bits are needed to

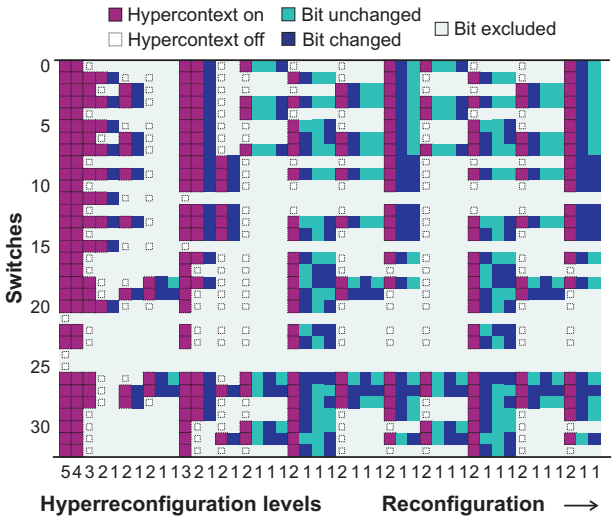

Figure 4. First 45 reconfigurations and 33 switches of the counter with 5 levels of reconfiguration

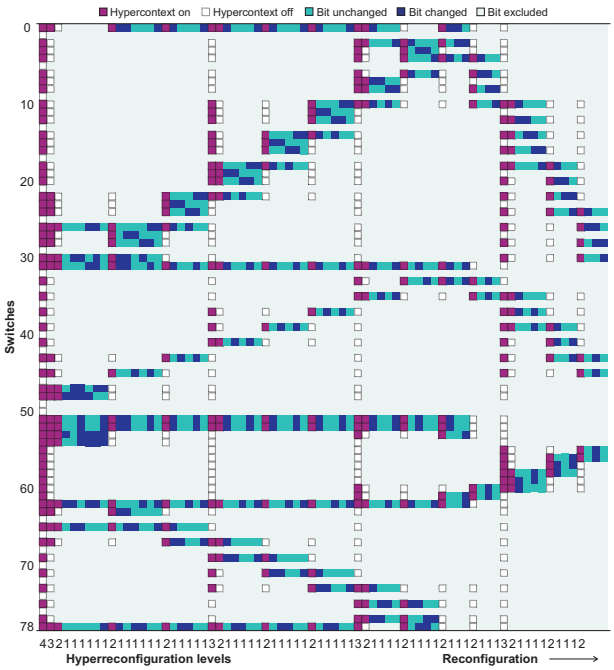

Figure 5. Adder with 4 levels of reconfiguration

define a context. The counter was executed 10 times which requires a total number of 110 reconfiguration operations. Figure 4 shows the first 45 reconfigurations for the first 33 switches of the optimal solution to the *Multi_phc* using 5 reconfiguration levels (Remark: the details are not relevant here, it is only important that we have the pattern of bit usages which results from the example circuit; for details on how to obtain the bit pattern see [3]). The reconfiguration data is shown from top (switch 0) to bottom (switch 32). The reconfiguration operations are drawn from left to right, starting with the highest reconfiguration level 5. Below the image the corresponding reconfiguration levels are given as numbers. The largest number of reconfigurations are performed on the lowest reconfiguration level 1. This level corresponds to ordinary reconfigurations known from traditional reconfigurable architectures.

The second example is an 8 bit ripple carry adder circuit on a SHyRA with 1 LUT, 1 MUX, 1 DeMUX, and 24 registers. 79 bits of reconfiguration data define a context, i.e., the function that is computed by the LUT and the routing that is realized by MUX and DeMUX. The resulting adder requires 56 reconfiguration operations. The solution of *multi_phc* for 4 reconfiguration levels is shown in Figure 5.

Test case 3 is an LED decoder which converts a 4 bit number to the 7 bit representation of a 7 segment LED display. This design was synthesized from a VHDL description and then mapped onto the SHyRA. Similar as for the adder 1 LUT, 1 MUX, and 1 DeMUX was used with 55 registers to store temporary results. The design resulted in 155 reconfiguration operations with 224 bits of reconfiguration data.

As the fourth test case a part of a simple control circuit is chosen and implemented on the SHyRA using 36 LUTs. Here only the reconfiguration data of the LUTs is taken into account. This way it is ensured that a large portion of the reconfigurable resources is active, because the routing resources do not change very frequently in most parts of the design. The resulting LUT reconfiguration data consists of 144 reconfiguration bits and 141 reconfiguration operations.

The optimal number of reconfiguration levels as well as the minimal reconfiguration costs are obtained by applying *Multi_phc* to architectures with 1 to $[m+1]$ reconfiguration levels, where m is the number of context requirements for the corresponding test design (see Table 1). This solves the RLP problem. Note that the cost reduction in the last column of the table shows the percentage by which the reconfiguration costs are reduced when compared to an architecture of only one reconfiguration level (i.e., only ordinary reconfiguration).

Figure 6 shows the optimal reconfiguration costs all test cases and 1-19 reconfiguration levels. In all cases costs decrease with higher number of reconfiguration levels until the optimal number of reconfiguration levels is reached. Then the costs increase with a higher

Table 1. Numbers of reconfigurations and switches, optimal number of reconfiguration levels and associated optimal costs for the test applications

Design	#Reconfigurations	#Switches	Optimal number of reconfiguration levels	Optimal cost	Cost reduction
8 bit RCA	56	79	3	1245	71.86%
Counter	110	48	2	3443	34.79%
LED Decoder	155	224	3	4527	86.96%
Control	141	144	2	10385	48.85%

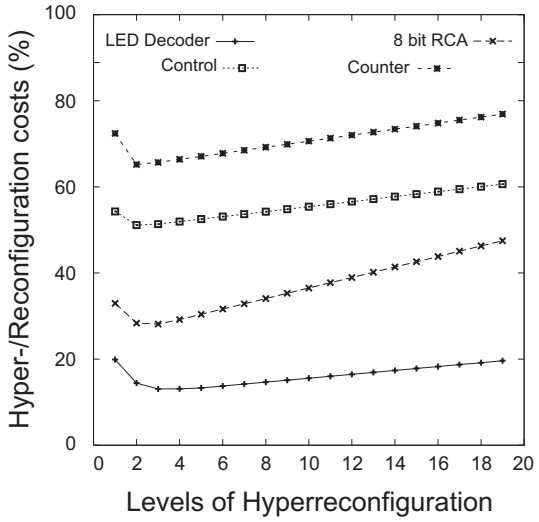

Figure 6. Optimal reconfiguration costs for different number of reconfiguration levels in percent of the costs for only one reconfiguration level

number of reconfiguration levels. Turning back to Figure 4 note that level 4 and 5 are only used once at the beginning and that the hypercontexts are the same in both cases. In this case the 4th-level is not very useful and increases only the reconfiguration costs. For analogous reasons the reconfiguration costs for the test cases increase linearly when the number of reconfiguration levels becomes large. In that case for each additional reconfiguration level one reconfiguration has to be done before the first computations. Each such reconfiguration has costs that are equal to the total number of switches used during the computation (in case that the reconfiguration are done according to an optimal solution of the PHC-Switch problem).

7. Conclusion

The concept of 2-level reconfigurable 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 multiple reconfiguration levels. We also introduced heterogeneous multi-level reconfigurable architectures where each switch can have a different number of reconfiguration levels. It was shown that optimal reconfigurations with an arbitrary number of reconfiguration levels can be found in polynomial time for the switch cost model. The problem of finding the optimal number of reconfiguration levels (RLP) was shown to be solvable in polynomial time on homogeneous multi-level architectures but it becomes NP-hard for heterogeneous multi-level architectures. experimental results for four applications (8 bit adder, counter, LED-decoder, and simple control circuit) on an architectures that has different reconfigurable units are presented. The results show that multiple reconfiguration level can lead to reduced reconfiguration costs.

References

[1] A. DeHon. DPGA utilization and application. In *FPGA '96: Proceedings of the 1996 ACM fourth international symposium on Field-programmable gate arrays*, pages 115–121, New York, NY, USA, 1996. ACM Press.

[2] S. Lange and M. Middendorf. Heuristics for context-caches in 2-level reconfigurable architectures. In *IEEE Conf. on Field-Programmable Technology (FPT' 05)*, 2005.

[3] S. Lange and M. Middendorf. Hyperreconfigurable architectures and the partition into hypercontexts problem. *J. Parallel Distrib. Comput.*, 65(6):743–754, 2005.

[4] S. Lange and M. Middendorf. On the design of two-level reconfigurable architectures. In *Int. Conf. on Rec. Comput. and FPGAs (ReConFig05)*, 2005.

[5] E. Mirsky and A. DeHon. MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In K. L. Pocek and J. Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 157–166, Los Alamitos, CA, 1996. IEEE Computer Society Press.