Task Scheduling for Context Minimization in Dynamically Reconfigurable Platforms

Nei-Chiung Perng and Shih-Hao Hung

Department of Computer Science and Information Engineering
National Taiwan Univeristy
106 Taipei, Taiwan
{d90011,hungsh}@csie.ntu.edu.tw

Abstract. Dynamically reconfigurable hardware provides useful means to reduce the time-to-prototype and even the time-to-market in product designs. It also offers a good alternative in reconfiguring hardware logics to optimize the system performance. This paper targets an essential issue in reconfigurable computing, i.e., the minimization of configuration contexts. We explore different constraints on the CONTEXT MINIMIZATION problem. When the resulting subproblems are polynomial-time solvable, optimal algorithms are presented.

1 Introduction

Dynamically reconfigurable hardware (DRH) allows partial reconfigurations to provide different functionalities over a limited number of hardware logics, compared to the popular Application-Specific Integrated Circuit (ASIC) approach. It was recently raised by many researchers that the DRH technology is very suitable to deal with the dynamism of multimedia applications [1, 2]. In such applications, instructions might be partitioned into coarse-grained tasks with a partial order and loaded onto dynamically reconfigurable devices for executions! Reconfigurability has recently become an important issue in the research community of embedded systems especially for FPGAs [3–6]. An FPGA configuration context, also referred to as a context, is the basic element to load a hardware description of a task. A multi-context system is a system with more than one FPGA chips or an FPGA chip with its configurable logic blocks being partitioned into several (equal-sized) areas. Although multi-context systems allow simultaneous task executions on different contexts, many implementations only allow the loading of one task at a time in reality [7].

One way to avoid the waiting time of task loadings is to pre-load proper hardware descriptions before the run time. In particular, Hauck [8] presented the concept of configuration prefetching in which the loading duration was overlapped with computations to reduce the overheads. Harkin et al. [9] evaluated nine approaches of hardware/software partitioning to provide insights in the implementation methodology for reconfiguration hardware. A genetic algorithm (GA) was also presented by the same authors for run-time reconfiguration [10]. Yuh et al. [11] developed a tree-based data structure, called T-tree, for a temporal floorplanning to schedule all the reconfigurable tasks with a simulated-annealing-based algorithm. Ghiasi et al. [12] proposed an efficient optimal algorithm to minimize the run-time reconfiguration delay in the executions of applications

on a dynamically adaptable system under assumptions on several restricted implementation constraints. Noguera and Badia [2] introduced a two-version dynamic scheduling algorithm for reconfigurable architectures with or without a prefetching unit. Resano et al. [13] proposed a way to revise a given task schedule by considering reconfiguration to minimize the latency overheads.

This paper targets one essential implementation issue in the reconfigurable computing: the minimization of the required number of FPGA configuration contexts, where the deadline and precedence constraints of an application are given. We consider the optimization problem without a given schedule (that comes with fixed execution intervals). The NP-completeness of the CONTEXT MINIMIZATION problem is first proved. Several additional constraints on the loading time and the execution time of a task and the precedence constraints of tasks (and their resulting subproblems) are explored. Optimal algorithms are presented for subproblems that are polynomial-time solvable. The objective is provide insights on how and why difficult the CONTEXT MINIMIZATION problem is.

The rest of this paper is organized as follows: Section 2 defines the CONTEXT MINIMIZATION problem. Section 3 explores several subproblems under several additional constraints. Optimal algorithms for subproblems that are solvable in polynomial time are presented. Section 4 is the conclusion.

2 Problem Definition

In this paper, we are interested in the derivation of a *reconfiguration plan* $\Re$ with the objective to minimize the number of required FPGA configuration contexts in a multi-context FPGA platform. The reconfiguration plan should be derived based on a given task set T, a partial order of task precedences $\prec$, and a common deadline D. Each task τ_i in a task set T is denoted as $\tau_i = (e_i, l_i)$, where e_i is the required execution time, and l_i is the loading (configuration) duration to load task τ_i onto a context. A precedence constraint $\tau_i \prec \tau_j$ in the partial order $\prec$ requires that task τ_j can only start its execution after the completion of task τ_i. A precedence constraint might exist because the latter task needs to read from the output of the former task. We are interested in the minimization of the maximum time span of the task execution in T. The problem is modeled as a performance requirement, i.e., the common deadline D. Any solution to the targeted problem is a reconfiguration plan $\Re$, in which we have a loading time $\Re_{\mathbb{T}}(\tau_i)$, an execution starting time $\Re_{\mathbb{S}}(\tau_i)$, and a configuration context ID $\Re_{\mathbb{C}}(\tau_i)$ for each task τ_i. A solution should also satisfy the given partial order of task executions and the common deadline. The problem is formally defined as follows.

Problem 1 (CONTEXT MINIMIZATION). Given a set T of n tasks ($\tau_i = (e_i, l_i) \in T, 1 \leq i \leq n$) with a partial order $\prec$ and a common deadline D, the problem is to find a reconfiguration plan $\Re$ with the minimum number of required FPGA configuration contexts without violating the partial order of task executions and the common deadline.

A reconfiguration plan is *feasible* if and only if the following three conditions are satisfied: The first condition requires each FPGA context being loaded in time. The second condition requires that any two tasks should not use the same context in any

overlapped time interval. The third condition requires the loading of contexts should be done one by one. The three conditions are defined formally as follows:

Condition 1 (In-Time Loading) $\forall \tau_i$, $\Re_{\mathbb{T}}(\tau_i) + l_i \leq \Re_{\mathbb{S}}(\tau_i)$ *and* $\Re_{\mathbb{S}}(\tau_i) + e_i \leq D$.

Condition 2 (Non-Overlapping Configuration Contexts) $\forall$ (τ_i, τ_j) *pair,* $\Re_{\mathbb{C}}(\tau_i) \neq \Re_{\mathbb{C}}(\tau_j)$ *if any two time intervals* $(\Re_{\mathbb{T}}(\tau_i), \Re_{\mathbb{S}}(\tau_i) + e_i]$ *and* $(\Re_{\mathbb{T}}(\tau_j), \Re_{\mathbb{S}}(\tau_j) + e_j]$ *have a non-null intersection.*

Condition 3 (Mutual Exclusion on Loading) $\forall$ (τ_i, τ_j) *pair,* $\Re_{\mathbb{T}}(\tau_i) + l_i \leq \Re_{\mathbb{T}}(\tau_j)$ *or* $\Re_{\mathbb{T}}(\tau_j) + l_j \leq \Re_{\mathbb{T}}(\tau_i)$.

A reconfiguration plan is *optimal* if it is feasible, and the number of its required contexts, i.e., the largest ID of the assigned contexts, is equal to the minimum number of required contexts of all feasible reconfiguration plans. We shall show later that this optimization problem is $\mathcal{NP}$-complete.

3 Problem Properties

In this section, we prove the $\mathcal{NP}$-completeness of the CONTEXT MINIMIZATION problem and later explore subproblems in which polynomial-time solutions exist. For the sake of clarity, we transform this optimization problem into an equivalent decision problem by providing a bound on the number of FPGA configuration contexts. The decision version of the CONTEXT MINIMIZATION problem is to find a solution with the number of required contexts no larger than a given number M.

3.1 $\mathcal{NP}$-Complete Subproblems

We shall show the $\mathcal{NP}$-completeness of two subprograms of the CONTEXT MINIMIZATION problem under two constraints: (1) The execution time of each task is identical, i.e., $\forall i$, $e_i = E$. (2) The loading duration of each task is negligible, i.e., $\forall i$, l_i = 0. Before we show the $\mathcal{NP}$-completeness of the two subproblems, we shall first define the PRECEDENCE CONSTRAINED SCHEDULING (SS9 in [14]), that is $\mathcal{NP}$-complete:

Given a set T of tasks, the execution time e_i of every task $\tau_i \in T$ is 1, a given number $M \in \mathcal{Z}^+$ of processors, a partial order $\prec$ of tasks $\in T$, and a deadline $D \in \mathcal{Z}^+$, the problem is to derive a schedule σ over the M processors so that the deadline and the partial order of task executions are satisfied. In other words, $\forall \tau_i, \tau_j \in T$, $\tau_i \prec \tau_j$ implies $\sigma(\tau_j) \geq \sigma(\tau_i) + e_i$, where $\sigma(\tau_i)$ denotes the starting time of task τ_i.

Theorem 1. *The CONTEXT MINIMIZATION problem under the constraint* $\forall i$, $e_i = E$ *and* $l_i = 0$ *is* $\mathcal{NP}$*-complete, where* e_i *and* l_i *denote the execution time and the loading duration of task* τ_i*, respectively.*

Proof. This subproblem is indeed the PRECEDENCE CONSTRAINED SCHEDULING problem. □

Theorem 2. *The CONTEXT MINIMIZATION problem under the constraint* $\forall i$, $l_i = 0$ *is* $\mathcal{NP}$*-complete, where* l_i *denotes the loading duration of task* τ_i.

Proof. The correctness of this theorem follows directly from the fact that the problem stated in Theorem 1 is a special case of this problem. □

Theorem 3. *The CONTEXT MINIMIZATION problem is* $\mathcal{NP}$*-complete.*

Proof. The correctness of this theorem follows directly from the fact that the problem stated in Theorem 2 is a special case of this problem. □

3.2 Subproblems in $\mathcal{P}$

This section is meant to explore three subproblems of the CONTEXT MINIMIZATION problem that have polynomial-time solutions. It is to gain the insight on why and how difficult the problem is.

A Task Set of Independent Tasks Although the CONTEXT MINIMIZATION problem is $\mathcal{NP}$-complete, the problem would become tractable under certain constraints. Suppose that all of the tasks are independent, i.e., $\prec = \emptyset$, and $\forall i$, $e_i = E$ and $l_i = L$ (i.e., the same execution time and loading duration for every task). The problem has optimal algorithms with a polynomial time complexity.

Algorithm 1

Input: A task set T ($\forall \tau_i \in T, \tau_i = (E, L)$), $\prec = \emptyset$, a deadline D, and M contexts
Output: A feasible reconfiguration plan $\Re$

1: $\Im = 0$.
2: **while** T is not empty **do**
3: Remove an arbitrary task τ_i from T.
4: Locate context M_j with the earliest idle time I_j.
5: $\Re_{\mathbb{T}}(\tau_i) = \max\{I_j, \Im\}$.
6: $\Re_{\mathbb{S}}(\tau_i) = \Re_{\mathbb{T}}(\tau_i) + L$.
7: $\Re_{\mathbb{C}}(\tau_i) = j$.
8: $\Im = \Re_{\mathbb{T}}(\tau_i) + l_i$.
9: **if** $\Re_{\mathbb{S}}(\tau_i) + E > D$ **then**
10: Return failure.
11: **end if**
12: **end while**

The subproblem is shown being $\mathcal{P}$-solvable by presenting a polynomial-time algorithm Algorithm 1: The input of the algorithm includes a task set, a common deadline, and an integer constant M that denotes the number of FPGA configuration contexts. Because of the mutual exclusion requirements on context loading (Condition 3), the algorithm maintains the earliest possible time for the next context loading, i.e., $\Im$. $\Im$ is initialized as 0 initially (Step 1). Each iteration of the loop between Step 2 and Step

12 is to schedule the loading of a task onto a context. The algorithm picks up a ready task arbitrarily (Step 3) and load the task onto the context with the earliest available time (Step 4). The loading time, starting time, and context ID of the task are updated accordingly (Steps 5-7). The earliest possible time for the next context loading is then updated (Please see Step 8 and Condition 3). The algorithm reports a failure if any task misses the deadline (Steps 9-11). The time complexity is $O(n \times \log M)$.

Theorem 4. *Algorithm 1 is optimal in the sense that it always derives a solution if any feasible solution exists.*

Proof. The correctness of this theorem follows directly from the fact that all tasks are of the same execution time and loading duration and share the common deadline. □

Figure 3.2 shows four optimal reconfiguration plans of four tasks, where the number M of FPGA configuration contexts ranges from 1 to 4. An interesting packing of tasks is shown in the figures, and the impacts of the mutual exclusion constraint, i.e., Condition 3, are clearly illustrated. Note that the shaded rectangles denote the loadings of tasks onto contexts, and white rectangles denote task executions.

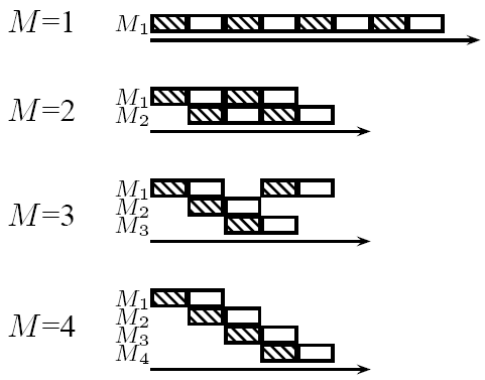

Fig. 1: Four reconfiguration plans over four different numbers of contexts

Lemma 1. *Algorithm 1 needs no more than* $M_B = \max\{n, \lceil \frac{E}{L} \rceil + 1\}$ *contexts to derive a feasible reconfiguration plan, where* n *is the number of tasks.*

Proof. The correctness of this lemma follows directly from the facts that loading durations can not be overlapped with each another, and a loading duration can be overlapped with any execution time as long as the three feasibility conditions are satisfied. □

Lemma 1 provides an upper bound on the maximum number of contexts over that Algorithm 1 could derive a feasible reconfiguration plan. Figure 2 shows reconfiguration plans for two task sets, i.e., one with $E \leq L$ and the other with $E > L$,

where different numbers of FPGA configuration contexts are tried. Note that when $n \times (L + E) \leq D$, only one context is needed to derive a feasible reconfiguration plan.

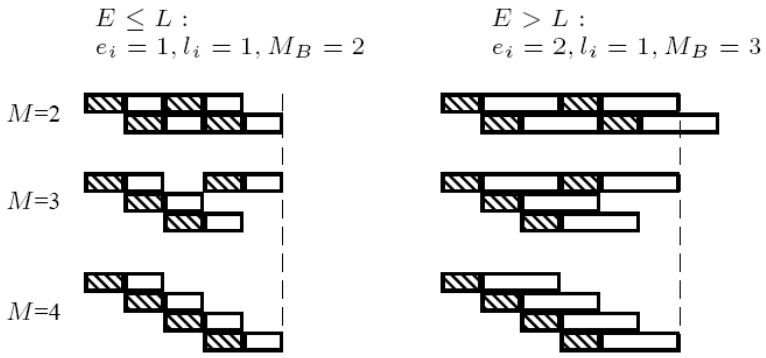

Fig. 2: Reconfiguration plans of two task sets over different numbers of contexts

Although we show that the CONTEXT MINIMIZATION problem becomes tractable when $\prec = \emptyset$, and $\forall i$, $e_i = E$ and $l_i = L$, one question remains. That is how difficult the problem is if we relax some of the above constraints! Suppose that we still have tasks being independent, i.e., $\prec = \emptyset$, but every task might have a different execution time, i.e., $e_i \neq e_j$ for some $\tau_i, \tau_j \in T$. Even if we let $\forall i$, $l_i = 0$, the CONTEXT MINIMIZATION problem is intractable because the problem is indeed the MINIMUM MULTIPROCESSOR SCHEDULING problem, which is a well-known $\mathcal{NP}$-complete problem [14].

A Task Set with a Tree Partial Order As shown in Theorem 1, the CONTEXT MINIMIZATION problem is $\mathcal{NP}$-complete when $\forall i$, $e_i = E$ and $l_i = 0$. However, the CONTEXT MINIMIZATION problem becomes tractable when all tasks are independent, i.e., $\prec = \emptyset$. In this section, we shall show that the CONTEXT MINIMIZATION problem remains tractable when $\prec$ is of a tree, and $\forall i$, $e_i = E$ and $l_i = 0$, regardless of whether it is an intree or an outtree (Figure 3).

We shall present an optimal algorithm based on the Critical Path (CP) rule [15]. A *critical path* of a partial order is defined as a path with the maximum number of tasks in a chain that follow the precedence constraints of the order. Note that when the completion time of a task on a critical path is delayed, the latest completion time of the tasks in the task set is delayed. Figure 3 shows an intree and an outtree. The path from the top-left node to the sink node of the intree in Figure 3 is an example critical path, and there are two critical paths in the intree. The CP rule provides a way to assign tasks priorities, where a task in the front of a critical path is assigned a higher priority, as shown in Figure 3. In fact, the priority assignment of tasks follows a topological order. Tie-breaking is done in an arbitrary way.

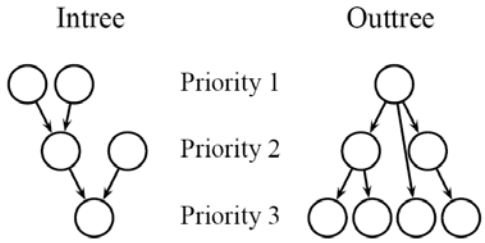

Fig. 3: An intree/outtree for some precedence constraints

Algorithm 2

Input: A task set T ($\forall \tau_i \in T, \tau_i = (E, 0)$), $\prec$ as a tree, a deadline D, and M contexts
Output: A feasible reconfiguration plan $\Re$

1: Assign priorities to tasks according to the CP rule.
2: S = { $\tau_i : \tau_i \in T$ does not have any predecessor }.
3: **while** S is not empty **do**
4: Remove the task τ_i with the highest priority from S.
5: Locate context M_j with the earliest idle time I_j.
6: $\Re_{\mathrm{T}}(\tau_i) = \Re_{\mathrm{S}}(\tau_i) = I_j$.
7: $\Re_{\mathrm{C}}(\tau_i) = j$.
8: $S = S \cup \{\tau_j\}$, where $\tau_i \prec \tau_j$ if all of the predecessors of τ_j have been scheduled.
9: **if** $\Re_{\mathrm{S}}(\tau_i) + E > D$ **then**
10: Return failure.
11: **end if**
12: **end while**

Algorithm 2 derives a feasible reconfiguration plan whenever possible: Tasks are first assigned priorities according to the CP rule (Step 1). Initially, S is set as the set of ready tasks in T (Step 2), where a ready task is a task with all of its preceding tasks in the partial order complete. Each iteration of the loop between Step 3 and Step 12 is to load a task onto a proper context. The ready task with the highest priority is loaded onto the context with the earliest possible idle time. The loading time and the starting time of the task is set as the earliest possible idle time of the context (Step 6), where $\forall i$, $l_i = 0$. The context ID of the task is then set (Step 7). After the task is scheduled, any ready task resulted from the scheduling join the ready task pool (Step 8). If the deadline is violated, then the algorithm reports a failure (Steps 9 and 10). The time complexity of Algorithm 2 is $O(n^2)$.

Theorem 5. *Algorithm 2 is optimal in the sense that it always derives a solution if any feasible solution exists.*

Proof. The optimality of the algorithm is based on the proof of the CP rule in [15]. □

A Task Set with $E \leq L$ As shown in the previous section, the CONTEXT MINIMIZATION problem becomes more tractable when a partial order is restricted in a tree

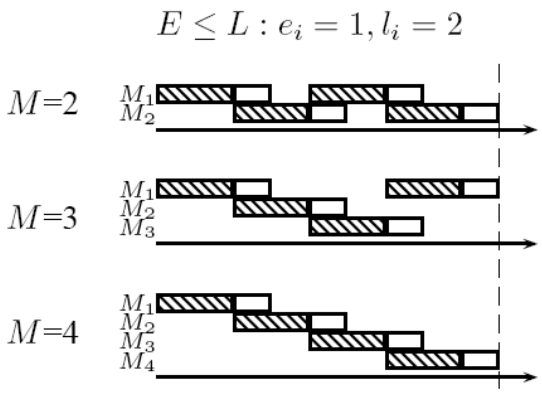

Fig. 4: Reconfiguration plans of a task set with $E \leq L$, where different numbers of contexts are used.

fashion, compared to that shown in Theorem 1. Another question is whether we could trade the partial-order constraint with any other constraint to keep the CONTEXT MINIMIZATION problem being tractable. In this section, we shall show that the CONTEXT MINIMIZATION problem remains tractable by the constraint $\forall i$, $e_i = E$, $l_i = L$, and $E \leq L$. In such a case, the partial order among tasks could be arbitrary.

An optimal algorithm for this problem is as the same as Algorithm 1 except two minor modifications: (1) Step 3: Remove any arbitrary ready task $\tau_i \in T$, and (2) Step 4: Use the context M_j which is idle in the last iteration. The algorithm is referred to as Algorithm 3. The time complexity of Algorithm 3 is $O(n)$.

Theorem 6. *Algorithm 3 is optimal in the sense that it always derives a solution if any feasible solution exists.*

Proof. The optimality of the algorithm is based on the fact that the minimum number of the required contexts must be 1 or 2, unless there is no feasible solution. If the summation of loading durations and execution times of all tasks is less than the common deadline, i.e., $n \times (L + E) \leq D$, the answer is 1; Otherwise, the answer is 2. □

Figure 4 illustrates the idea of the proposed algorithm and provides further explanation of the above theorem. As shown in the figure, there are three reconfiguration plans of a task set derived by Algorithm 3, where the number of contexts ranges from 2 to 4. Because of the mutual exclusion requirements on context loadings (i.e., Condition 3), only two contexts are needed to load tasks in turn. The execution time of each task on one context is contained in the loading duration of another task on the other context, where these two tasks are scheduled one after another. We shall point out that the CONTEXT MINIMIZATION problem would become much more intractable when constraints of Section 3 are relaxed.

4 Conclusion

In this paper, we explore different constraints on the CONTEXT MINIMIZATION problem, and provide the complexity proofs of $\mathcal{NP}$-complete subproblems and the optimal algorithms of subproblems in $\mathcal{P}$. We are currently investigating heuristic-based greedy algorithms for the CONTEXT MINIMIZATION problem. We will further explore the problem on the reconfigurable platforms allowing one-dimension allocation.

References

[1] Kneip, J., Schmale, B., Moller, H.: Applying and implementing the mpeg-4 multimedia standard. IEEE Micro **19**(6) (November-December 1999) 64–74

[2] Noguera, J., Badia, R.M.: Multitasking on reconfigurable architectures: Microarchitecture support and dynamic scheduling. ACM Transactions on Embedded Computing Systems **3**(2) (May 2004) 385–406

[3] De Micheli, G., Gupta, R.K.: Hardware/software co-design. Proceedings of the IEEE **85**(3) (March 1997) 349–365

[4] DeHon, A., Wawrzynek, J.: Reconfigurable computing: What, why, and implicatios for design automation. In: Proceedings of the 36th ACM/IEEE Conference on Design Automation. (1999) 610–615

[5] Hauck, S.: The roles of FPGA's in reprogrammable systems. Proceedings of IEEE **86**(4) (April 1998)

[6] Wolf, W.: FPGA-Based System Design. Prentice Hall (2004)

[7] Xilinx Inc.: XAPP151 Virtex Series Configuration Architecture User Guide. (v1.7) edn. (October 2004)

[8] Hauck, S.: Configuration prefetch for single context reconfigurable coprocessors. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (1998)

[9] Harkin, J., McGinnity, T.M., Maguire, L.P.: Partitioning methodology for dynamically reconfigurable embedded systems. IEE Proceedings on Computers and Digital Techniques **147**(6) (November 2000) 391–396

[10] Harkin, J., McGinnity, T.M., Maguire, L.P.: Modeling and optimizing run-time reconfiguration using evolutionary computation. ACM Transactions on Embedded Computing Systems **3**(4) (November 2004) 661–685

[11] Yuh, P.H., Yang, C.L., Chang, Y.W.: Temporal floorplanning using the T-tree formulation. In: Proceedings of ACM/IEEE International Conference on Computer-Aided Design. (2004)

[12] Ghiasi, S., Nahapetian, A., Sarrafzadeh, M.: An optimal algorithm for minimizing run-time reconfiguration delay. ACM Transactions on Embedded Computing Systems **3**(2) (May 2004) 237–256

[13] Resano, J., Mozos, D., Catthoor, F.: A reconfigurable manager for dynamically reconfigurable hardware. IEEE Design and Test of Computers **22**(5) (2005)

[14] Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Company (1979)

[15] Pinedo, M.: Scheduling Theory, Algorithms, and Systems. second edn. Prentice Hall (2002)