A Task Graph Approach for Efficient Exploitation of Reconfiguration in Dynamically Reconfigurable Systems

Kyprianos Papademetriou and Apostolos Dollas *
Department of Electronic and Computer Engineering
Technical University of Crete
GR73100 Chania, Crete, Greece
$\{$*kpapadim, dollas*$\}$*@mhl.tuc.gr*

1 Introduction

Partial reconfiguration suffers from the inherent high latency and low throughput which is more considerable when reconfiguration is performed on-demand. This work deals with this overhead in processors combining a fixed processing unit (FPU), and a reconfigurable processing unit (RPU). Static and dynamic prefetching [1], and instruction forecasting [2] are targeting at reduction of the overhead through preloading of configurations. Banerjee et al. [3] transform the task graph of an application and a heuristic algorithm evaluates the reduction in schedule length and selects the most promising configuration. Tasks are scheduled according to the physical resource constraints. In our work we augment the prefetching model of [1] by taking into account the hardware area constraints of a partially reconfigurable system. Given the task graph of an application, tasks with low probability to be executed are split and preloaded according to the hardware in order to be fully utilized. Thus, the time during which reconfiguration is overlapped with processor execution is increased.

2 Proposed Model

We describe the problem and the proposed approach with an illustrative example. Figure 1(a) has the graph of an application wherein each task represents an RPU operation (RPUOP). In t0, among other instructions, a decision is made regarding which one of t1, t2 should be followed. If all three tasks can be simultaneously placed onto the RPU, no delay is incurred during transition from t0 to the next task. On the contrary, if a resource-constrained RPU is employed that can at most hold t0 and one of the t1, t2 e.g., $S(t0, t1) < A < S(t0, t1, t2)$, where S is the total size of the tasks in parentheses and A is the size of the available hardware, a delay might be incurred. Only tasks that statically fit in the chip are prefetched according to the static prefetching model presented in [1]. Assuming that t1 is the task most likely to be executed, its prefetch is placed before t0 as shown in Figure 1(a). If the outcome of t0 needs t2, the execution is stalled waiting for the FPU to partially reconfigure the RPU; this is performed with the prefetch of t2.

We augment the original model with a mechanism that statically determines the best prefetching, which utilizes all the physical resources. This is illustrated in Figure 1(b). Assume that the size of the available hardware is $A = S(t0, t1, t2 \times 2/3)$. By performing transformations on the task graph, t2 is split into two subtasks. Subtask t2a is $2/3\times$ the t2 size and the remaining $1/3$ is the size of t2b. Subtasks t2a and t2b are then processed as two different tasks. The only factor that is changed for each subtask is configuration latency (L) which is proportional to the area. The t2a is prefetched as it fits the available hardware. In Figure 1 configuration latency is expressed in quantities of time, rather than in time units. Assuming that t2 is needed, the original model incurs a configuration latency equal to 60. In the augmented model only t2b is needed to be loaded which entails a latency of 20.

3 Experiments and Discussion

A reconfigurable processor and a simple application were modelled with TGFF [4]. In the generated graph of Figure 2, tasks t0_0, t0_1 and t0_3 are executed by the FPU whereas tasks t0_2 and t0_4 are RPUOPs. The latter can not coexist in the RPU due to its restricted area.

Fig 2 (a) illustrates the prefetch insertions for the original model. In scenario A, t0_2 is prefetched before t0_0. If t0_2 is to be executed no new prefetch is needed. If t0_4 is to be executed the next prefetch is needed incurring a greater reconfiguration overhead than the former case does. Figure 2(b) has the prefetches for the augmented model. In scenario A t0_2 and t0_4a are prefetched. If t0_4 is to be executed the prefetch for t0_4b is called. The incurred overhead in this case is investigated in our experiments. We consider whether reconfiguration of RPUOPs can overlap with the tasks executed in the FPU i.e., t0_3 for scenario A or t0_1 for scenario B.

A limited testbench is constructed consisting of 20 sys-

*Also at ITRI, Wright State University, Dayton, Ohio, USA

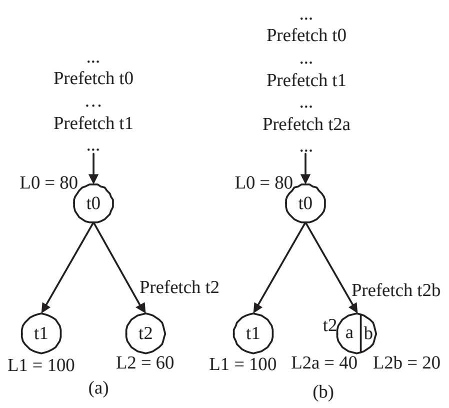

Figure 1. (a) Initial graph and prefetches according to the original static prefetching model. (b) Transformed graph and prefetches according to the augmented model.

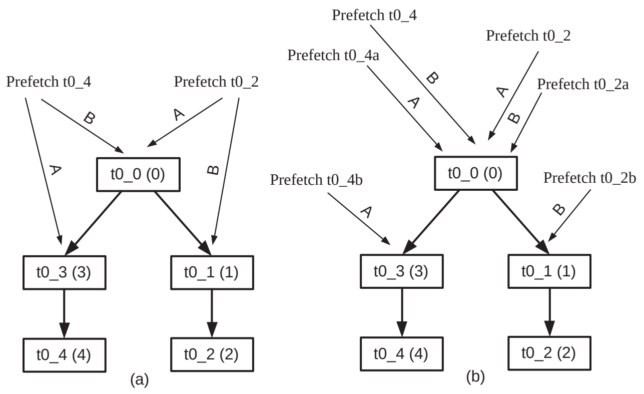

Figure 2. Prefetches according to the original (a) and the augmented model (b). Two different scenarios regading the insertion of prefetches are shown by the labels A and B.

tems which are randomly generated by TGFF for which the structure of the task graph is common. Regarding the FPU it is assumed that each task takes an execution of $100 \pm 80\mu s$. The RPU roughly resembles a Virtex-II XC2V40 device with 64 CLBs. It is assumed that CLBs are the only resources used for the application. The average number of CLBs required by a task is chosen to 30 ± 25. Configuration latency is computed based on Xilinx's data sheets. Virtex-II devices are partially reconfigurable and the frame is the smallest unit of reconfiguration. A pad frame is added at the end of the configuration data. The XC2V40 has 404 frames and an 8-bit ICAP interface running at $66MHz$ between the FPU and the RPU. Therefore, the time needed to configure all frames is $0.642ms$ whereas the time for one frame is $1.58\mu s$ (pad frame not included). For sake of simplicity configuration latency per CLB instead of frame is used. 176 frames are needed to configure all CLBs which gives 2.75 frames per CLB. The time needed to configure one CLB is $(1 \times 2.75 \times 1.58) + 1.58 = 5.92\mu s$, for two CLBs $(2 \times 2.75 \times 1.58) + 1.58 = 10.27\mu s$ etc.

Table 1 has some results and the average of all experiments regarding the reconfiguration overhead, i.e., the amount of time that can not be hidden by overlapping reconfiguration with processor execution. ROOM and ROAM are the Reconfiguration Overhead for the Original Model and the Augmented Model respectively, caused by the loading of the prefetch after the branch. The positive and negative numbers represent the configuration time that can/can't be hidden by processor execution respectively. In some cases the augmented model completely hides configuration time, and in all cases it performed better than the original model.

4 Conclusions and Future Work

In the future, we will evaluate the model with real data. Moreover, we plan to incorporate communication and memory issues to our experimental framework. Also, we will investigate how transformations should affect the split of a task i.e., parallelizable functional subtasks vs. non-functional subtasks that are reassembled for execution.

Table 1. Experimental results for the two models.

	E0	E1	E9	E15	E19	Avg
ROOM	-193.4	-179.8	-100.8	-160.4	-377.3	-236.1
ROAM	-38.7	68.3	54.7	108.5	-243.3	-59.6

5 Acknowledgements

We thank the Greek Ministry of National Education and Religious Affairs for supporting Kyprianos Papademetriou with a Ph.D fellowship under the program Heraklitus, grant number 88727/11. We also thank Prof. Nikolaos Bourbakis, Director of the ITRI Institute at Wright State University for hosting Prof. Dollas during his sabbatical.

References

[1] Z. Li, "Configuration Management Techniques for Reconfigurable Computing," PhD thesis, Northwestern University, 2002.

[2] M. Iliopoulos and T. Antonakopoulos, "Run-Time Optimized Reconfiguation Using Instruction Forecasting," in *Field Programmable Logic and Applications*, 2001.

[3] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Considering Run-Time Reconfiguration Overhead in Task Graph Transformations for Dynamically Reconfigurable Architectures," in *Field Programmable Custom Computing Machines*, 2005.

[4] R. Dick, D. Rhodes, and W. Wolf, "TGFF: Task Graphs For Free," in *Proceedings of the International Workshop on Hardware/Software Codesign*, 1998.