Temporal Task Clustering for Online Placement on Reconfigurable Hardware

Ali Ahmadinia Christophe Bobda Jürgen Teich
Department of Computer Science 12, University of Erlangen-Nuremberg, Germany
{ahmadinia, bobda, teich}@cs.fau.de

Abstract

Partial reconfiguration allows for mapping and executing several tasks on an FPGA during run time. One of the challenging problems in multitasking systems is high amount of communication costs. In this paper, we present two clustering methodologies that temporally cluster real-time tasks for a partially reconfigurable hardware and reduce communication overhead. The first algorithm aims at efficient use of resources by clustering close run-time tasks, and the second one makes the clustering with respect to a trade off between inter-task communication and resource utilization efficiency. The results show significant reduction in communication costs.

1. Introduction

Recent generations of FPGAs allow run-time partial reconfiguration, which gives the ability for configuring a task onto an FPGA without affecting other currently running tasks. While this technique can increase device utilization, it also increases communication between reconfigurable hardware and host processor, especially when we have more inter-task communication.

The reconfigurable system is usually an FPGA which is connected to a host processor by a bus. As the bus width is limited, the communication has to be time multiplexed on the bus if many data is transported on the processor-FPGA bus [2]. This happens e.g. when a module has to be replaced in the FPGA. Because the communication between the tasks is done by inter configuration register set inside the FPGA. All the temporary data in those registers have to be saved in the processor's address space before reconfiguration. The device is then reconfigured and the data are copied back into the FPGA registers [3]. Then we should minimize the register set for communicating of tasks, to decrease communication between FPGA and processor. One of the best approaches to achieve this goal is clustering. Because when a cluster is placed on FPGA, a number of tasks will be configured consecutively. Then the communication time for configuration will be decreased. In addition, we can combine interdependent tasks for execution on the same cluster in order to reduce communication costs, since the communication will be done internally. Also clustering decreases the complexity of the scheduling algorithm. The rest of this paper is organized as follows: Section 2 provides the background and algorithms of task clustering. In Section 3 the used online placement will be described. Experimental results and conclusion will be presented in section 4 and 5 respectively.

2. Task Clustering

2.1. Background and Definitions

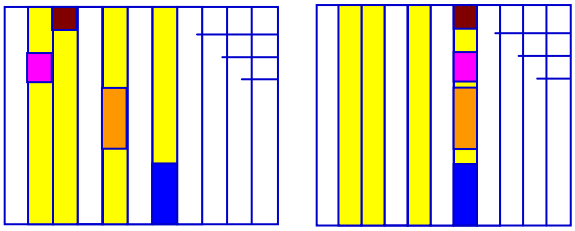

Figure 1. The influence of clustering on reconfiguration overhead

Currently Xilinx FPGA technology (Virtex series) can be partially reconfigured by changing only frames (Figure 1). As we know reconfiguration will be done by exchanging frames [1]. We aim at minimizing the number of frames to be reconfigured and avoid interrupting the other running modules in the same columns. This goal can be reached if modules which

should be reconfigured at the same time are placed in consecutive frames. This requires a clustering approach for a given set of tasks. For this reason we must define our task and cluster models.

Definition 1 (Task Characteristics) Given a set of tasks $T= \{ t_1 ,t_2 , \ldots, t_m \}$ such that,

$\forall t_k \in T, \quad t_k = (a_k, e_k, d_k, area_k)$

a_k = arrival time of task t_k
e_k = execution time of task t_k
d_k = deadline time of task t_k
$area_k$ = area of task t_k

A clustering $C= \{ C_1,C_2 , \ldots,C_n \}$ is a partition of $T= \{ t_1 ,t_2 , \ldots, t_m \}$ into disjoint subsets $C_1 ,C_2 , \ldots, C_n$ of T.

Definition 2 (Cluster Characteristics) Mapping tasks from set T to the clustering set $C= \{ C_1 ,C_2 , \ldots, C_n \}$.

$$\forall C_i \in C, \quad C_i = (start_i, finish_i, ALAP_i, ASAP\_end_i, ALAP\_end_i, m_i)$$

$start_i$ = start running time of cluster C_i
$finish_i$ = finish running time of cluster C_i
$ALAP_i$ = the latest possible time to start execution of cluster C_i
$ASAP\_end_i$ = the earliest possible time from the current time to finish execution of cluster C_i
$ALAP\_end_i$ = the latest possible time to finish execution of cluster C_i
m_i = the number of tasks in cluster C_i

A cluster is a set of components to be placed at the same time on the device. We should define the criteria for optimizing the task clustering. One of the important parameters in task clustering is wasted area of resources. Figure 2 shows the time intervals that the area occupied by task t_i in cluster c_k is wasted.

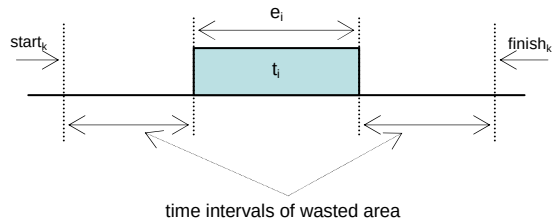

Figure 2. The wasted area due to cluster-based placement

Definition 3 (Wasted Area) This parameter shows how long and how much area is occupied but not used:

$$Wasted\_Area = \sum_{k=1}^{n} \sum_{i=1}^{m_k} (finish_k - start_k - e_i) \times area_i$$

Wasted area is influenced by running time of tasks, then the running time is one of the factors for optimizing the clustering. For being able to free the space occupied by the components of a cluster, those components should have nearly the same run-time.

If we want to assign a task to cluster, we must be sure that the task can be executed at the start running time of the cluster. Then the start running time of the cluster must be in the mobility interval of task. It means that $start_i \in [ASAP(t_i), ALAP(t_i)]$. Also, the task must have been finished at the same time of cluster end time. Since usually the tasks have not exactly the same execution time, we use a margin parameter (δ) to make the end time more flexible. This parameter can be optimized according to the cost functions. Now we can define the clustering criteria as follows:

Definition 4 (Clustering Criteria) A task t_k that is placed in the cluster C_i will satisfy the following conditions:

1) $a_k \le start_i \le d_k - e_k$ 2) $finish_i - start_i - e_k \le \delta$

Another criterion that must be considered is inter-task communication. As we place the tasks of one cluster at the same time, and obviously in adjacent positions, then it would be much easier to setup task communication inside a cluster than having more connections between clusters.

Definition 5 (Task Communication) Given a set of communication task W,

$W = \{ w_{ij} | i = 1,..,m \ \ and \ \ j = 1,..,m \}$

$\forall t_i, t_j \in V, w_{ij} \in W$

Then there is a communication from t_i to t_j with bus width of w_{ij}.

As we mentioned before we should minimize the inter-configuration register set for communicating between clusters, then we can define this cost function:

Definition 6 (Inter_Cluster Communication) This cost function shows how much data exchange should be managed between clusters. The amount of communication from cluster C_k to cluster C_j is $\sum_{i=1}^{m_k} \sum_{l=1}^{m_j} w_{il}$. Therefore the total communication is :

$$Inter\_Cluster \ \ Communication = \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{m_k} \sum_{l=1}^{m_j} w_{il}$$

We can use communication factor for clustering, but as we know communication exists between dependent tasks. Now we can say it is worth to put dependent task in a cluster, only if both tasks have short running time, and also occupy a small area or when they have large width of communication. Otherwise we waste a large area of reconfigurable hardware for a long time. Therefore we should make a trade off between inter-task communication and efficient use of resource:

Definition 7 (Communication Trade off) We have a communication between task t_i and t_j with bus width of

w_{ij} (t_i => t_j) and w_{ji} (t_j => t_i). If we put task t_i and t_j in one cluster we have the following cost function:

$$Communication\ \ Trade\ \ off(CT_{ij}) = \alpha \times Communication\_Width + \frac{\beta}{\mathrm{Re}source\_Wasting}$$

$$Communication\_Width = w_{ij} + w_{ji}$$

$$\mathrm{Re}source\_Wasting = area_j \times e_i \ \ + \ \ area_i \times e_j$$

In the above equation, *Communication_Width* shows the amount of communication between two tasks, and the *Resource_Wasting* parameter defines how long and how much area will be wasted. Factor α captures the significance of communication, and factor β captures the importance of wasting resources. We can change these two factors to optimize the clustering.

2.2. Algorithms

As explained above, we have proposed two criteria for clustering, but it is not possible to combine these two methods because their properties are against each other. Therefore two algorithms: 1. Close Run Time Clustering and 2.Communication Trade off Clustering have been developed.

As we told when we have a communication from task t_i to task t_j , it means that task t_j is dependent on task t_i, and task t_j can be executed until task t_i is finished. Consequently we assume that the deadline time of task t_i is earlier than arrival time of task t_j. Therefore in our clustering algorithms we don't have to worry about dependencies of tasks, as they have been considered in the deadline and arrival times of tasks.

First the clustering construction algorithm, based on the running time of tasks will be described. As each task arrives, it will be put in an existing cluster or a new cluster will be created for the task depending on the characteristics of the task and the clusters. The clustering set *C* and tasks set *T* in the beginning are empty, *k* and *i* are the numbers of created clusters and arrived tasks respectively:

Algorithm 1. Close Run Time Clustering

C_P1: *for j=1 to k // Cluster placing*
if ((ALAP_end$_j$–ASAP_end$_j$–current_time)< ε)
{Place C_j on the reconfigurable device.
Delete C_j from clustering set.
start$_j$ = current time;
finish$_j$ = ASAP_end$_j$ + current_time;}
if(task_request(t)) Go to ***Clustering1****;*
else Go to ***C_P1****;*

Clustering1*:i=i+1;*
t_i <= t;
feasible_cluster = 0;
for j=1 to k
if((a_i< ALAP$_j$)and(| ASAP_end$_j$ – e_i|< δ))
{Put t_i in C_j.
ALAP$_j$= min(d_i - e_i , ALAP$_j$);
ASAP_end$_j$= max(e_i , ASAP_end$_j$);
ALAP_end$_j$= min(d_i , ALAP_end$_j$);
feasible_cluster = 1;
break;}
if (feasible_cluster = 0)
{Create_new_Cluster();
k=k+1;
Put t_i in C_k.
ALAP$_k$= d_i - e_i ;
ASAP_end$_k$= e_i ;
ALAP_end$_k$= d_i ;}
Go to ***C_P1****;*

Parameter ε represents the minimum time that is needed to initialize running the cluster on the device. The second algorithm is based on the communication of tasks. The task will put in a cluster which its tasks have the maximum *Communication Trade off* with it:

Algorithm 2. Communication Trade off Clustering

C_P2: *for j=1 to k // Cluster placing*
if ((ALAP_end$_j$–ASAP_end$_j$–current_time)< ε)
{Place C_j on the reconfigurable device.
Delete C_j from clustering set.
start$_j$ = current time;
finish$_j$ = ASAP_end$_j$ + current_time;}
if(task_request(t)) Go to ***Clustering2****;*
else Go to ***C_P2****;*

Clustering2*:task_request(t_i);*
feasible_cluster = 0;
max=0;
CT_{max}=0;
for j=1 to k
if(a_i < ALAP$_j$)
{feasible_cluster = 1;
CT_j=0;
for l = 1 to m_j
CT_j= CT_j+ CT_{li}+ CT_{il};
if CT_j> CT_{max}
{CT_{max}= CT_j;
max=j;}}
if (feasible_cluster = 1)
{Put t_i in C_{max}.
ALAP$_{max}$= min(d_i - e_i , ALAP$_{max}$);
ASAP_end$_{max}$= max(e_i , ASAP_end$_{max}$);
ALAP_end$_{max}$= min(d_i , ALAP_end$_{max}$);}
else
{Create_new_Cluster();
k=k+1;
Put t_i in C_k.
ALAP$_k$= d_i - e_i ;
ASAP_end$_k$= e_i ;
ALAP_end$_k$= d_i ;}
Go to ***C_P2****;*

In the above algorithm, m_j is the number of tasks in cluster C_j, and CT_j is the summation of Communication Trade-off of the arriving task t_i with existing task in the cluster C_j.

3. Online Cluster Placement

After constructing clusters, the next step is to determine where to place a cluster on the reconfigurable hardware. This decision determines the fragmentation of the reconfigurable surface. A high fragmentation can lead to the undesirable situation that a cluster can not be placed although there would be sufficient area available. Therefore an efficient online placement algorithm should be used to avoid such situations. Existing online placement algorithms assume that the template has a rectangular shape. Here we also assume when we cluster a set of tasks, then a rectangle shaped cluster will be created according to the area of its tasks.

There are two main parts in online placement algorithms. The first part is handling empty rectangles. In [4], two categories of methods have been implemented: 1.Keeping all the maximal empty rectangles (KAMER). 2.Keeping disjoint empty rectangles. Since the first method keeps all the maximal empty rectangles and checks all of them for placing an arriving module, the quality of its placement is better than any method the second category. The second part of placement is similar to online bin-packing problem. They [4] have proposed two-dimensional extensions of First Fit(FF), Best Fit(BF) and Bottom Left(BL).

Here, we apply the KAMER method [4], which has not the best performance, but the best quality. Also we use FF strategy, but this is not important since we want to evaluate our algorithms against our cost functions.

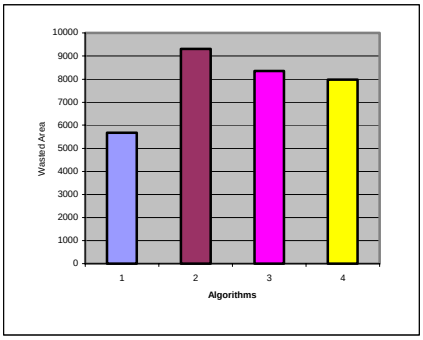

Figure 3. Comparison of wasted area for clustering algorithms

4. Experimental Results

To investigate the influence of our proposed methods, we have implemented a system model, with randomly generated task sets. 50 task sets have been generated with 100 tasks each. Simulation results have been averaged over these 50 task sets. Figures 3,4 show the results of four clustering algorithms against the cost functions. These four algorithms are as follows:

1. Close Run-time Clustering
2. Communication Trade-off Clustering *(α =10, β =1)*
3. Communication Trade-off Clustering *(α =5, β =5)*
4. Communication Trade-off Clustering *(α =1, β =10)*

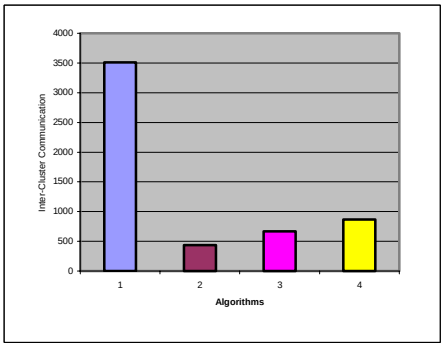

Figure 4. Comparison of inter-cluster communication for clustering algorithms

5. Conclusion

In this paper, we have proposed two clustering algorithms to reduce the communication costs. The first algorithm is based on close running time of tasks, and the second one aims at reducing communication between clusters by using a trade-off function for inter-task communication and the area will be wasted because of clustering interdependent tasks.

The results show that the first algorithm uses resources efficiently, but gives a large amount of communication between clusters. The second method wastes resources more but reduces the communication cost considerably, and also it can be used to tune communication cost and resource utilization based on system characteristics.

6. References

[1] D. Mesquita, F. Moraes, J. Palma. L. Möller, and N. Calazans. “Remote and Partial Reconfiguration of FPGA: tools and trends”. In Proc. RAW Workshop. IEEE-CS Press, April 2003.

[2] H. Kalte, M. Porrmann, and U. Rückert. “A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs”. In Proc. of Heterogeneous reconfigurable SoC Workshop, Germany, April 2002.

[3] C. Bobda. “Temporal Partitioning and Sequencing of Dataflow Graphs on Reconfigurable Systems”. In Proc. of DIPES World Congress, Canada, 2002, pages 185-194.

[4] K. Bazargan, R. Kastner, and M. Sarrafzadeh. “Fast Template Placement for Reconfigurable Computing Systems”. In IEEE Design and Test of Computers, volume 17, 2000, pages 68-83.