Journal of Engineering and Applied Sciences 3 (10): 766-773, 2008
ISSN: 1816-949X

A Novel Temporal Partitioning Algorithm for Run Time Reconfigured Systems

[1]Bouraoui Ouni, [1]Ramzi Ayadi and [2]Mohamed Abid
[1]Laboratory (E.μ.M), Faculty of Sciences at Monastir, 5000, Monastir, Tunisia
[2]C.E.S, National Engineering School of Sfax, (ENIS), B.P.W. 3038, Sfax, Tunisia

Abstract: This study focuses on introducing a new temporal partitioning algorithm. It divides the input task graph model into an optimal number of partitions and puts each task in the appropriate partition in order to decrease the transfer of data required between partitions of the design. However, typical scheduling algorithms focus on minimizing the overall latency of an input target graph.

Key words: Temporal partitioning, reconfigurable computing system, algorithm architecture adequacy

INTRODUCTION

The reconfiguration capability of the Run Time Reconfigured (RTR) systems can be used to perform a large application by partitioning the application over time into multiple segments. The division of an application into temporal segments that are configured one after the one is called temporal partitioning. The first temporal partition receives input data, performs computations and stores the intermediate data into the on-board memory. The device is then reconfigured for the next segment, which computes results based on the intermediate data, from the previous partition. This process is repeated until all the partitions are executed. The RTR system consists of a reconfigurable area (usually based on dynamically reconfigurable FPGA) communicating with an external memory. Each temporal partitioning is mapped to the reconfigurable area and the data flowing between two temporal partitions is mapped to the memory. A Host interacts with both the reconfigurable area and the memory and it used to load new configurations and to store the intermediate data in the memory. In this study, we consider that we have an available reconfigurable hardware area which should be exploited for the implementation of a given algorithm. This area constraint presents a problem, which may be caused by global consideration related to the system and the application constraints (e.g., when it requires more than available reconfigurable area). Or for the fact that the system is already existed and no extensions are possible, whereas adding a new service or a modification in the existing application is required. The typical idea consists in partitioning the application over time into multiple segments. In the temporal partitioning field, the main objective of related works is either to find the minimal size of reconfigurable area to accomplish the graph within a fixed limit of time or to find the minimal execution time of the input graph on a fixed-size of area (Kaul and Vermuri, 1999; Cardoso and Neto, 1999; Heng and Ronald, 2005; Bobda, 2003, 2007; Chang and Sadowska, 1999). However, the proposed approach focuses on minimizing the required data transfer between different temporal partitions of the design. Thus, the main objective of the study, consists in introducing a temporal partitioning algorithm which divides the input task graph model to an optimal number of time partitions and puts each task in the appropriate partition in order to decrease the transfer of data between partitions.

DEFINITIONS

Task graph: Considering a set of tasks V = {T1, T2,…., Tk}, a task graph model Ouaiss *et al.* (1998a) is defined by the couple

$$G = (V; E)$$

where:

V = The nods set
E = The arcs set

An arc, e = (Ti,Tj) ∈ E, defines a dependence of data between task Ti task and the task Tj. According to this model, a task is an abstraction of a behavioural algorithm that should be implanted in a same partition.

Task parameters: Considering a set of tasks V = (T1, T2, … Tn), the characteristics of a task Ti are:

A (Ti): The area occupation (surface) in CLB of the task Ti. This surface can be gotten by using a synthesis tools (Kaul *et al.*, 1998; Xu and Kurdahi, 1998).

Corresponding Author: Bouraoui Ouni, Laboratory (E.μ.M), Faculty of Sciences at Monastir, 5000, Monastir, Tunisia

L (Ti): The time of execution (latency) of the Ti task. This parameter can be also gotten by using a synthesis tools.

Arc parameters: An arc e = (Ti; Tj) ∈ E, noted by Ti → Tj defines the dependence of data between the task Ti and the task Tj. The arc characteristics are:

B (Ti,Tj): The quantity of data that should be transferred from the task Ti toward the task Tj.

L (eij): The necessary time to transmit a quantity of data from given from the task Ti toward the task Tj.

Temporal partitioning: The temporal partitioning is the division, under one or several constraints, of a graph G (V, E) to a set of disjointed partitions P = {P1, P2... "Pn"} such as:

$$\cup_{k=1}^{n} P_K = P$$

where, $\forall P_K \in P$, we have

$$\sum_{Ti \in P_K} A(Ti) \leq R\max$$

A (Ti) represents the area of the task Ti, R_{max} represents the area constraint.

Connectivity: The connectivity (Con (G)) of a given graph G (V, E) is the relation of number of edges in G over the number of all edges which can be built with the nodes of G.

$$Con\ (G) = (2N_E) / (N_V\ (N_V\text{-}1))$$

where:
N_E = The number of edges
N_V = The number of nodes in G

Quality: Given a graph G and a set of time partitions $\{P_0, P_1 \ldots P_n\}$, the quality (Q (G)) of the graph G is calculated as follows:

$$Q(G) = \frac{1}{N}\sum_{i=1}^{i=Np} Con(Pi);$$

N is the number of partition

The quality is a good way to evaluate the performance of a given temporal partitioning algorithm in term of data transfer. Indeed, if the algorithm assigns dependent tasks to a same partition, then the quality will be high and the communication cost will be low. The algorithm performs well because it minimizes the set of memorization resources used to save data during the reconfiguration of the reconfigurable area. However, if the algorithm assigns dependent tasks to different partitions, then the quality will be low and the communication cost will be high. The algorithm would be bad because it maximizes the set of memorization resources used to save data during the reconfiguration of the reconfigurable area. In Fig. 1, we present an illustrative example, without taking care of any constraint, of two temporal partitioning. In the Fig. 1a the result of temporal partitioning uses seven access (red arrows) to the external memory. However, in the Fig. 1b the result of temporal partitioning uses only three access to the external memory. Therefore, we favor the second algorithm if we have a major constraint on the external memory.

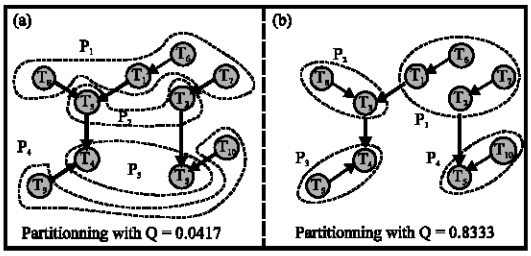

Fig. 1: Example of two temporal partitioning

CONSTRAINTS

The behavior specifications are in form of task graph. For each task, High Level Synthesis (HLS) tool generated the synthesis cost in terms of the resource requirement and execution delay of task. Typically, a temporal partitioning problem is developed under these constraints.

R_{max}: The resource capacity of the reconfigurable area: The sum of resource cost of all the tasks mapped to a temporal partition must be less than the overall resource constraint of the available reconfigurable area.

Precedence constraint: This constraint can be defined as follows: If a task T2 depends on a task T1. Then the last should be placed the first.

Uniqueness constraint: According to this constraint, every task should be placed in a unique partition.

M_{max}: The temporary on-board memory: Data transfer across partition boundaries will occur due to two dependent tasks being placed in deferent temporal

partition. This intermediate data needs to be stored between partitions and should be less than the memory constraint, M_{max}, of the reconfigurable device.

RELATED WORKS

In the literature, early researches in synthesis field solved the spatial partitioning problem (Cupta and Demicheli, 1990; Abid, 2000). With the introduction of reconfigurable systems several approaches are interesting in this new field. These studies are interested in solving some problems that appear during the design process for reconfigurable systems such as the temporal partitioning problem.

ILP technique: The "ILP" Kaul *et al.* (1998) and Byungil (1999) is one of the most approaches that used to solve the temporal partitioning problem. This technique introduces a variable "Y_{Tp}", where "Y_{Tp}" =1 if the task "T" is placed in partition p, otherwise "Y_{Tp}" = 0. The inputs of the "ILP" approach are: The lower bound and the upper bound on partitions number. Memory and area constraints. The dependencies between tasks. The number of data unit communicates between tasks "T_i" and "T_j" and the area of each task. The aim of the "ILP" approach consists in finding the optimal solution in term of latency for the temporal partitioning problem, while satisfying imposed constraints. The main limitation of the "ILP" approach is its very high execution time, which hardly gives solutions for a graph which has several nodes.

List scheduling technique: The list scheduling algorithm is used by Cardoso and Neto (1999), Chang and Sadowska (1998) and Puna and Bahitia (1999) to solve the temporal partitioning problem. The main idea of this method consists in placing all nodes of the graph on a list. The first partition is built by removing nodes from the list to the partition until the size of the target FPGA is reached. Then, a new partition is built and the process is repeated until all nodes are placed in partition. This technique is often oriented by the ASAP and/or the ALAP scheduling. In Pandey and Vemuri (1999) the authors put a node in the list as soon as all predecessors have been placed in the list. However, in Ouaiss *et al.* (1998b) the authors place a node in the list as soon as all successors have been placed in the list. The main advantage of the list scheduling technique is the very fast run time of its algorithm (Cardoso and Neto, 1999; Bobda, 2003). However, the performance of this technique is lower than the performance of the Chang and Sadowska (1999), Ouni (2008) and Bobda (2003).

Dynamic algorithm: The dynamic algorithm Ouni *et al.* (2005) is composed by two main steps: Finding the minimal number of partitions (N_{min}) and the maximal number of partition (N_{max}). Realizing a schedule of the tasks in this minimal number of partition in order to reduce to the latency of the application. The principle steps of this algorithm are presented in Fig. 2. This algorithm is a good candidate for the temporal partitioning problem, but its main limitation is its high execution time.

Proposed temporal partitioning algorithm: As we remark the typical scheduling algorithms in this field are focused only on minimizing the overall latency of the design and do not consider the memory as a major constraint. For this reason, in these techniques, each partition includes several parallel tasks. So, if the exchanged data between partitions is important, the memory constraint becomes difficult to reach before building several partitions or adding other resources (memories, registers) to handle the shared data between different partitions. Hence, the introduction of new temporal partitioning algorithms, as presented in this section, which focus on minimizing the data transfer required between different time partitions of the design, represents a very important issue in computer aided design of Reconfigurable Computing Systems. The proposed algorithm divides the input task graph model to an optimal number of partitions and puts each task in the appropriate partition in order to decrease the communication cost between design partitions. To attain this aim, we should maximize dependent tasks in each partition. For this reason we built a dependency list for each node (line 3). Then our algorithm puts a note in target partition and it removes notes from its dependency list (line 12) to this partition until the size of the available reconfigurable area is reached. We should note that the algorithm is based on this hypothesis:

$$\forall\, T_i \in (1, N_V);\ \text{we have } A(T_i) \leq A_c$$

where:

$A(T_j)$ = The area of task T_j
Nv = The number of task in G
A_c = The area constraint

Now, we provide the description of our proposed temporal partitioning algorithm. First, all nodes of the graph are inserted according to their priorities on a List-nodes (line 1). In this algorithm we follow steps presented by David (1999) and Chang and Sadwska (1999) to calculate the priority of each node in the task graph model.

Then the algorithm builds for each node "ni" a List_dependency_node"ni" (line 3). This list is calculated as follow:

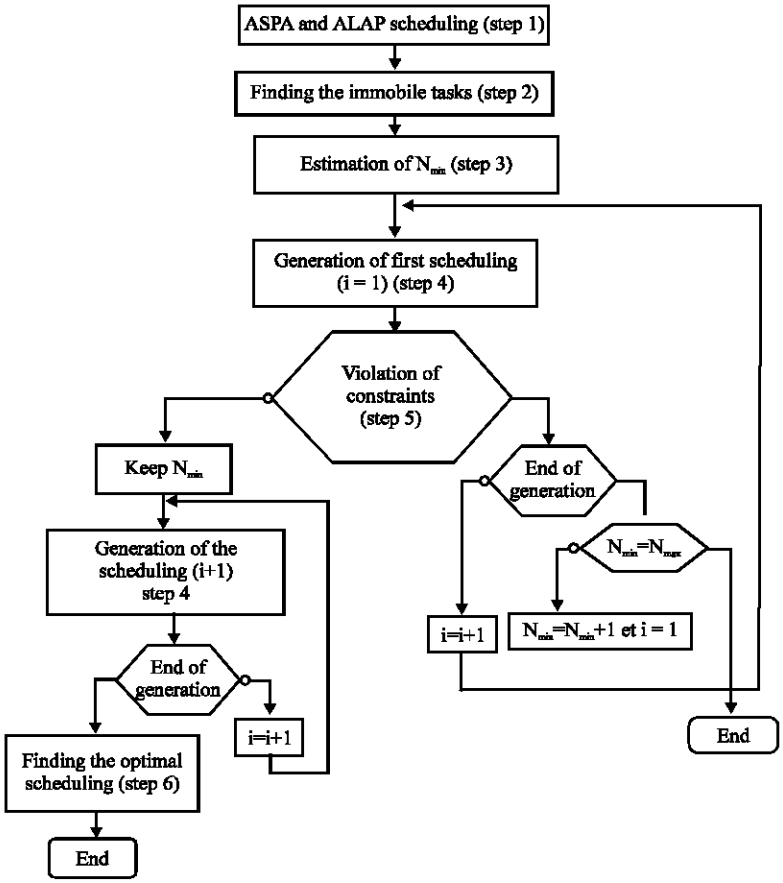

Fig. 2: Dynamic algorithm

$$\text{Dep\_list (ni)} = \left\{ \phi_{ij}\ n_j \right\}_{j=1\ \text{To}\ V}$$

where, $\Phi_{i,j}$ =1 if node "n_j" depends on node "n_i"; otherwise $\Phi_{i,j}$ = 0.

Next the algorithm puts the first node "ni" from the List-nodes in the first partition (4, 6, 7 and line 8). Then the algorithm displaces nodes from the List_ dependency_node"ni" to this partition until the size of the available reconfigurable area is reached (9, 10, 11, 12, 13 and line 14). Next the algorithm inserts this partition in a List_patitions (line 15). If there is at least one node in the List-nodes (line 5) then the algorithm repeats the previous process (6, 7, 8, 9, 10, 11, 12, 13, 14 and line 15).

CASE STUDY

The Color Layout Description "CLD" is a low-level visual descriptor that can be extracted from images or video frames. The process of the CLD extraction consists of four stages, the image partitioning, the selection of a single representative color for each block, the DCT transformation and the non linear quantization and Zig Zag scanning (Kaul *et al*., 1998). Since, DCT is the most computationally intensive part of the CLD algorithm, it was often chosen to be implemented in hardware and the rest of subtasks (partitioning, color selection, quantization, zig-zag scanning and Huffman encoding) were often chosen for software implementation. The model proposed by Kaul *et al.* (1998) is based on 16 vector products. Thus, the entire DCT is a collection of 16 tasks, where each task is a vector product. There are 2 kinds of tasks in the task graph proposed by Kaul *et al.* (1998), "T_1" and "T_2", whose structure is similar to vector product, but whose bit widths differ.

In this study, we use four algorithms in order to divide each of 16 task graph. We use the ILP algorithm, the list scheduling algorithm, the dynamic algorithm and the proposed algorithm. In each case, we evaluate the performance of each algorithm in terms of quality in terms of words stored in each partition and in terms of whole latency. In this study, we only considered the latency (execution time) of tasks to calculate the whole latency of the design. We not considered the time needed for communication between the design partitions. The 16

DCT task graph, shown in Fig. 3, is chosen to be implemented within several available reconfigurable areas. We note that we used our framework to draw those graphs (Ouni *et al*., 2004; Ouni, 2008). We associate three parameters for each task, the first parameter its occupation area, the second is its latency, the third is its energy consummation. The later is usually equal zero because we disregard it in this study. The area and the computation time of each task shown in Fig. 4 are taken from Kaul *et al*. (1998). The Table 1 and Fig. 5 give the different solutions provided by the list scheduling, the "ILP" technique, the dynamic algorithm and the proposed algorithm for the temporal partitioning problem. For the illustrated examples, results show that our algorithm decreases very better the communication cost than the others algorithms. Indeed, the quality given by our algorithm is too greater than the quality given by others technique. And the number of words stored in each partition is always the lowest. Thus, our algorithm can be qualified to be a good candidate if designers aim to reduce the communication cost in the design.

```
1:      List-nodes = generate-node_list_with priority (G)
2:      For (i=1; i<NV;i++)
                {
3:        List_dependency_node(ni) = generate
List_dependency_node(ni) with priority (G)
                }
4:      k=1
5:      While (list_ nodes≠Φ)
                {
6:        Node (ni) = first no removed node from List_node
7:       Partition (k) <= (node (i))
8:      Size_ partition (k) = size_node(ni)
9:       (next-node) <=0
10:       Next_node_size = 0
11:       While (size_partition(k)+Next_node.size()≤  Ac )
                {
12:         Partition (k) <= (next_node);
13:        (next-node) <= first no removed node from
List_dependency_node(ni)
14:         Next_node.size=size of first no removed node from
 List_dependency_node(ni)
                }
15:       List_patitions<=partition (k);
16:     k = k+1
                }
```

Fig. 3: Proposed algorithm

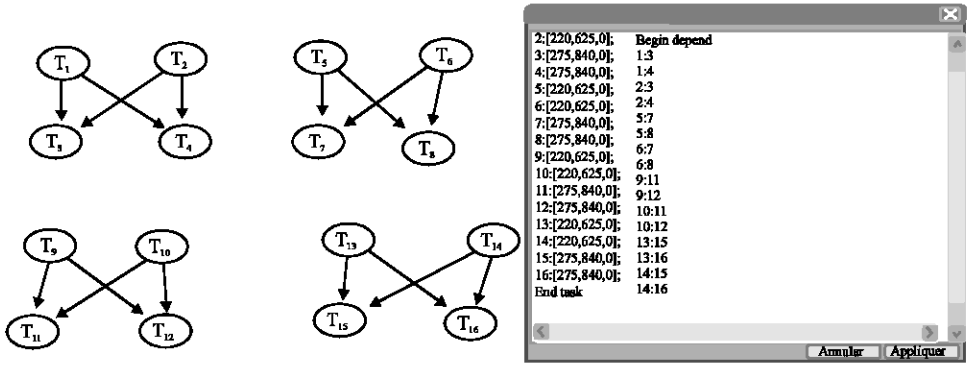

Fig. 4: DCT task graph

Table 1: Design result

	List scheduling	Proposed algorithm	ILP algorithm	Dynamic algorithm
Area (CLB)	1200	1200	1200	1200
Number of partitions	4	4	4	4
Tasks in partition 1	T1,T2,T5,T6,T9	T1,T2,T3,T4	T1, T2, T5, T6,	T1,T2,T5,T6,T9
Words stored in partition 1	10 words	2 words	8 words	10 words
Tasks in partition 2	T3, T10, T13, T14	T5, T6, T7, T8	T9, T10, T13, T14	T10, T13, T14
Words stored in partition 2	7 words	2 words	8 words	6 words
Tasks in partition 3	T4, T7, T8, T11	T9, T10, T11, T12	T3, T4, T7, T8,	T3, T4, T7, T8,
Words stored in partition 3	4 words	2 words	4 words	4 words
Task in partition 4	T12, T15, T16	T13, T14, T15, T16	T11,T12, T15 , T16	T11,T12, T15 , T16
Words stored in partition 4	3 words	2 words	4 words	4 words
Whole latency	3145+3* C_T	5860+3* C_T	2930+3* C_T	2930+3* C_T
Quality	0	0,66	0	0
Area (CLB)	1400	1400	1400	1400
Number of partitions	3	3	3	3
Tasks in partition 1	T1, T2, T4, T5, T6	T1,T2,T3,T4	T1, T2, T5, T6,	T1,T2,T5,T6,T9
Words stored in partition 1	7 words	2 words	8 words	10 words
Tasks in partition 2	T3, T7, T8, T9, T10	T5, T6, T7, T8	T9, T10, T13, T14	T10, T13, T14
Words stored in partition 2	7 words	2 words	8 words	6 words

Table 1: Continued

	List scheduling	Proposed algorithm	ILP algorithm	Dynamic algorithm
Tasks in partition 3	T11, T12, T 13, T14	T9, T10, T11, T12	T3, T4, T7, T8,	T3, T4, T7, T8,
Words stored in partition 3	6 words	2 words	4 words	4 words
Number of partitions	T16, T15	T13, T14, T15, T16	T11,T12, T15 , T16	T11,T12, T15 , T16
Quality	0,05	0,66	0	0
Whole latency	3985* C_T	5860+3* C_T	2930+3* C_T	2930+3* C_T
Area (CLB)	1600	1600	1600	1600
Number of partitions	3	3	3	3
Tasks in partition 1	T1,T2,T5,T6 ,T9, T10, T3	T1,T2,T3,T4 ,T5, T6	T1,T2, T5, T6, T9,T10, T13	T1,T2, T5, T6, T9,T10, T13
Words stored in partition 1	11 words	6 words	14 words	14 words
Tasks in partition 2	T13,T14,T4,T7, T8, T15	T7,T8,T9,T10, T11, T12,	T14, T3, T4, T7, T8, T11	T14, T11, T3, T4, T7, T8,
Words stored in partition 2	6 words	4 words	7 words	7 words
Tasks in partition 3	T11, T12, T16	T13, T14, T15, T16	T12, T15, T16	T12, T15, T16
Words stored in partition 3	3 words	2 words	3 words	3 words
Whole latency	3770 ns+ 3* C_T	4395ns + 3* C_T	2305 +3* C_T	2305 +3* C_T
Quality	0,07	0,39	0	0
Area (CLB)	2050	2050	2050	2050
Number of partitions	2	2	2	2
Tasks in partition 1	T1, T2, T4, T5, T6, T9, T10, T13, T14	T1,T2,T3,T4, T5, T6, T7, T8	T1, T2, T5, T6, T9, T10, T13, T14, T3	T5, T6, T9, T10, T13, T14, T1, T2, T3
Words stored in partition 1	15 words	4 words	15 words	15 words
Tasks in partition 2	T3, T7, T8, T11, T12, T15, T16	T9, T10, T11, T12, T13, T14, T15, T16	T4, T7, T8, T11, T12, T15 , T16	T11,T12, T15 , T16, T4, T7, T8,
Words stored in partition 2	7 words	4 words	7 words	7 words
Whole latency	2305+2* C_T	2930+2* C_T	2305+2* C_T	2305+2* C_T
Quality	0,02	0,28	0,02	0,02
Area (CLB)	2400	2400	2400	2400
Number of partitions	2	2	2	2
Tasks in partition 1	T1, T2, T3, T4, T5, T6, T7, T8, T9	T1, T2, T3, T4, T5, T6, T7, T8	T1, T2, T5, T6, T9, T10, T13, T14, T3, T4	T5, T6, T9, T10, T13, T14, T1, T2, T3, T4
Tasks in partition 2	T10, T11, T12, T13, T14, T15, T16	T9, T10, T11, T12, T13, T14, T15, T16	T7, T8, T11, T12, T15, T16	T11, T12, T15, T16, T7, T8,
Whole latency	2930+2* C_T	2930+2* C_T	2305+2* C_T	2305+2* C_T
Quality	0,25	0,28	0,04	0,04
Area (CLB)	2500	2500	2500	2500
Number of partitions	2	2	2	2
Tasks in partition 1	T1, T2, T3, T4, T5, T6, T7, T8, T9, T10	T1, T2, T3, T4, T5, T6, T7, T8	T1, T2, T5, T6, T9, T10, T13, T14, T3, T4	T5, T6, T9, T10, T13, T14, T1, T2, T3, T4
Words stored in partition 1	8 words	4 words	14 words	14 words
Tasks in partition 2	T11, T12, T13, T14, T15, T16	T9, T10, T11, T12, T13, T14, T15, T16	T7, T8, T11, T12, T15, T16	T11, T12, T15, T16, T7, T8,
Words stored in partition 2	4 words	4 words	6 words	6 words
Whole latency	2930+2* C_T	2930+2* C_T	2305+2* C_T	2305+2* C_T
Quality	0,22	0,28	0,04	0,04
Area (CLB)	2800	2800	2800	2800
Number of partitions	2	2	2	2
Tasks in partition 1	T1, T2, T3, T4, T5, T6, T7, T8, T9 T10, T13	T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11	T1, T2, T5, T6, T9, T10, T13, T14, T3, T4 , T7,	T5, T6, T9, T10, T13, T14, T1, T2, T3, T4, T7
Words stored in partition 1	8 words	7 words	13 words	1 3 words
Tasks in partition 2	T14, T11, T12, , T15, T16	T12, T13, T14, T15, T16	T8, T11, T12, T15, T16	T11, T12, T15, T16, T8
Words stored in partition 2	6 words	3 words	5 words	5 words
Whole latency	2930+2* C_T	2930+2* C_T	2305+2* C_T	2305+2* C_T
Quality	0,20	0 ,29	0,05	0,05
Area (CLB)	3000	3000	3000	3000
Number of partitions	2	2	2	2
Tasks in partition 1	T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12	T1, T2, T3, T4, T5, T6, T7, T8 T9, T10, T11, T12	T1, T2, T5, T6, T9, T10, T13, T14, T3, T4, T7, T8	T5, T6, T9, T10, T13, T14, T1, T2, T3, T4, T7, T8
Words stored in partition 1	6 words	6 words	12 words	12 words
Tasks in partition 2	T13, T14, T15, T16	T13, T14, T15, T16	T11, T12, T15, T16	T11, T12, T15, T16
Words stored in partition 2	2 words	2 words	4 words	4 words
Whole latency	2930+2* C_T	2930+2* C_T	2305+2* C_T	2305+2* C_T
Quality	0,42	0,42	0,06	0,06

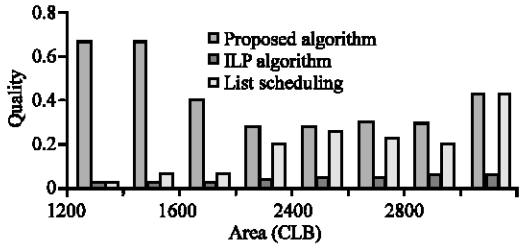

Fig. 5: Quality = f (area)

CONCLUSION

In this study, we have investigated the temporal partitioning used for reconfigurable systems. A new algorithm is introduced; it has the advantage of being able to divide the input task graph onto set of time partitions while decreasing communication cost between the design partitions. However, previous works interested in the temporal partitioning field were introduced algorithms, such as ILP, List scheduling, network flow, which divide the input graph while minimizing the whole latency of target application without considering the memory constraint. To better illustrate the efficiency of our technique, we devote a part of this paper to implement some practical examples, such as DCT task graph, by using our temporal partitioning algorithm. The studied evaluation cases show that the proposed algorithm provides very significant results in terms communication cost versus other well known algorithms used in the temporal partitioning field.

REFERENCES

Abid, M., 2000. Contribution for mixed software/hardware electronic systems design, Thesis of state, National school Engineers of Tunis, Tunisia.

Bobda, C., 2007. Introduction to Reconfigurable Computing Architectures, Algorithms and Applications, Springer Publishers. ISBN: 978-1-4020-6088-5 (HB), (e-book), pp: 362.

Bobda, C., 2003. Synthesis of Dataflow Graphs for Reconfigurable Systems using Temporal Partitioning and Temporal Placement, Thesis 2003, Faculty of Computer Science, Electrical Engineering and Mathematics of the University of Paderborn Germany.

Byungil, J., 1999. Hardware software partitioning for reconfigurable architectures, MS Theses School of Electrical Engineering, Seoul National University.

Cardoso, J.M.P. and H.C. Neto, 1999. An enhance static-list scheduling algorithm for temporal partitioning onto rues. IFIP TC10 WG10.5 10 Int. Conf. Very Large Scale Integration(VLSI'99), Portugal, pp: 485-496.

Chang, D. and M. Sadowska, 1999. Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs. IEEE Trans. Comput., 48 (6) 565-578.

Chang, D. and M. Sadowska, 1998. Partitioning sequential circuits on dynamically reconfigurable FPGAs, International Symposium on Field Programmable Gate Arrays (FPGA 98), Monterey, California, pp: 161-167.

Cupta, R. and G. Demicheli, 1990. Partitioning of functional models of synchronous digital systems. Computer-Aided Design, ICCAD, Digest of Technical Papers IEEE International Conferenceon Stanford University, CA, USA., 77: 216-219. ISBN: 0-8186-2055-2.

David, R., 1999. Synthese de la transformation en ondelette par l'outil monet, mastre these preparee au sein de laboratoire LESTER France, soutenu le 23 juin.

Heng, T. and F.D. Ronald, 2005. A Device-Controlled Dynamic Configuration Framework Supporting Heterogeneous Resource Management. In: Proc. Eng. Reconfigurable Syst. Algorithms (ERSA 2005), Las Vegas, pp: 251-254.

Kaul, K. and R. Vermuri, 1999. Integrate Block processing and design space exploration in temporal partitioning for RTR architecture. In: Reconfigurable architecture workshop, RAW'99. Springer Publication, pp: 606-615.

Kaul, K., R. Vermuri, S. Govindarajan and I. Ouaiss, 1998. An automated temporal partitioning tool for a class of DSP application, workshop and reconfigurable computing in international conference on parallel architecture and compilation technique PACT, pp: 22-27

Ouaiss, I., S. Govindarajan and V. Srinivasan, 1998a. A unified specification model for concurrency and coordination for synthesis VHDL, international conferences on information systems, analysis and synthesis ISAS 1998, pp: 771-778.

Ouaiss, I., S. Govindrajan, V. Srinivasan, K. Kaul and R. Vermuri, 1998b. An integrated partitioning and synthesis system for dynamically reconfigurable multi-FPGA architectures, IPPS/SPDP workshops, pp: 31-36.

Ouni, B., 2008. Synthese et partitionnement temporel pour les systemes reconfigurables, These universitaire, Faculte des Sciences de Monastir, Tunisie, Fevrier.

Ouni, B., A. Mtibaa and M. Abid, 2004. Temporal Partitioning Framework for fully Reconfigurable Systems. The 16th Int. IEEE Conf. Microelectronics, ICM, Gammart, Tunisia, pp: 742-745.

Ouni, B., A. Mtibaa and M. Abid, 2005. Synthesis and time partitioning for reconfigurable system. Design Automation for Embedded Syst. J., 9: 177-191.

Pandey, A. and R. Vemuri, 1999. Combined temporal partitioning and scheduling for reconfigurable architectures technology FPGA for computing and application. Proceeding of the SPIE 3844, pp: 93-103. DOI: 10.1117/12.359528.

Puna, K.M.G. and D. Bahitia, 1999. Temporal partitioning and scheduling data flow graphs for reconfigurable computers. IEEE Trans. Comput., 48 (6): 579-590.

Xu, M. and F. Kurdahi, 1998. Layout driven high level synthesis for FPGAs Based architecture. Proceedings of the Conference on Design, automation and test in Europe. IEEE Computer Society Washington DL. USA., pp: 446-450. ISBN: 0-8186-8359-7.