Mapping of Massive Data Processing Systems to FPGA Computers Based on Temporal Partitioning and Design Space Exploration

Paulo Sérgio Brandão do Nascimento[1,2], Stelita M. da Silva[1], Jordana L. Seixas[1], Remy E. Sant'Anna[1,2], Manoel E. de Lima[1]

[1] Centro de Informática – UFPE – Caixa Postal 7851 Cidade Universitária, Fone +55 81 2126.8430 Fax: +55 81 2126.8438 Recife – PE – Brazil
[2] Centro Federal de Educação Tecnológica de Pernambuco – CEFET-PE, Av. Prof. Luiz Freire – 500, Cidade Universitária, CEP: 50740-740 Fone: +55 81 2125.1716 Recife – PE – Brazil
e-mail: {psbn, sms, jls, res, mel}@cin.ufpe.br

ABSTRACT

High parallelism degree is fundamental for high speed massive data processing systems. Modern FPGA devices can provide such parallelism plus flexibility. However, these devices are still limited by their logic block size, memory size, memory bandwidth and configuration time. Temporal partitioning techniques can be a solution for such problems when FPGAs are used to implement large systems. In this case, the system is split into partitions (called contexts), multiplexed in a FPGA, by using reconfiguration techniques. This approach can increase the effective area for system implementation, allowing increase of parallelism in each task that composes the application. However, the necessary reconfiguration time between contexts can cause performance decrease. A possible solution for this is an intensive parallelism exploration of massive data application to compensate for this overhead and improve global performance. This is true for modern FPGA with relatively high reconfiguration speed. In this work, A reconfigurable computer platform and design space exploration techniques are proposed for mapping of such massive data applications, as image processing, in FPGA devices, depending on the application task scheduling. A library with different hardware implementation for a different parallelism degree is used for better adjustment of space/time for each task. Experiments demonstrate the efficiency of this approach when compared to the optimal mapping reached by exhaustive timing search in the complete design space exploration. A design flow is shown based on library components that implements typical tasks used in the domain of applications.

Index Terms: FPGA-Computers, Massive Processing, Temporal Partitioning, Design Space Exploration, Area-Time Trade-offs.

1. INTRODUCTION

Massive data applications, as image analyses or high precise numeric model simulations, in general, require a high performance computation approach [1][3][4][10]. Therefore, architectures to face such kind of problems should be able to provide such resources and its intrinsic parallelism. Nowadays, reconfigurable devices like powerful Field Programmable Gate Arrays (FPGA) [2], appear as a good possible solution for such high speed kind of problem. However, FPGAs are also limited by their logic resources, memory size and bandwidth, that define amount of data that can be read and processed simultaneously. The focus in this work is applications composed of massive data tasks as image tasks, vector-radix-fft, convolution, digital filters, median filter, image edge detection, morphologic transformations (erosion, dilation) and others tasks used in the large numeric data structure analyzer systems [11].

The approach considers that a given application is composed of a set of dependent tasks $A=\{Tsk_1, Tsk_2,...,Tsk_N\}$ represented by a dataflow (*DFG*). For each task Tsk_i a set of possible RTL implementation $I\_set(Tsk_i)$ is associated. These sets constitute a library used in the design. These sets are generated by hardware design experts with high level synthesis tools [9]. As depicted in Figure 1, each $I\_set(Tsk_i)$ represents

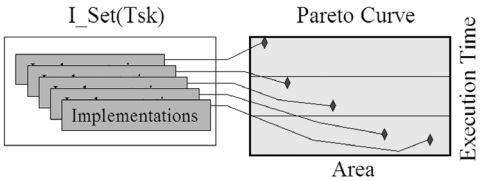


Figure 1. Design space of each task: *Area* ×*Time* pareto curve.

the *Area×Time* design space of tasks, where each task implementation is one point in the pareto curve [12].

By definition, each point in the pareto curve is the best solution, relative each other points, in at least one parameter (area or time) while being at least the same in the other parameter [12]. In this approach, if the designer generates a solution that it is not on the pareto curve, its implementation is discarded.

An application $\boldsymbol{A}$ can be split into temporal partitions $\boldsymbol{\Pi} = \{\boldsymbol{\pi}_1, \boldsymbol{\pi}_2,...,\boldsymbol{\pi}_K\}$, where each partition $\boldsymbol{\pi}_a \subset \boldsymbol{A}$. These partitions, or contexts, are sequentially executed by a FPGA temporal multiplexing mechanism based on hardware reconfiguration [5]. This multiplexing mechanism, also called hardware virtualization or multi-context, is useful when a given FPGA area is not enough to support the total application logic. The increase in effective area generated by the temporary partitioning allows the use of optimal execution time implementation of each task on pareto curves, because these implementations need more area for parallelism increase (Figure 1). In spite of additional reconfiguration time overhead, appropriate number of contexts and appropriate choice of task implementations can improve the global performance of the application. The two main problems are: how to define the temporal partitioning mapping and to choose the best task implementation, from components in the library, that guarantee the required performance of an application.

These problems are closely related. A heuristic to simultaneous resolution of these problems is suggested in function of FPGA area constraint and the application execution time constraint.

The next section presents the reconfigurable platform architecture proposed in this work. Section 3 presents related works. The heuristic design flow is presented in details in Section 4. In Section 5 experimental results are discussed. Finally, Section 6 presents conclusions and future works.

2. RECONFIGURABLE PLATFORM ARCHITECTURE

The platform architecture is depicted in Figure 2. The platform host is based on Nios II Soft-Core System in a Stratix FPGA architecture, from Altera, running μCLinux operating system [6].

The host is responsible for all system management, context reconfiguration, tasks execution and input and output of data in the applications. In the Stratix board a compact flash card is used to keep the context configurations. Data and configurations are manipulated in the host as μCLinux OS files. This allows the use of standard OS services for the platform software control (*ReconfSw*).

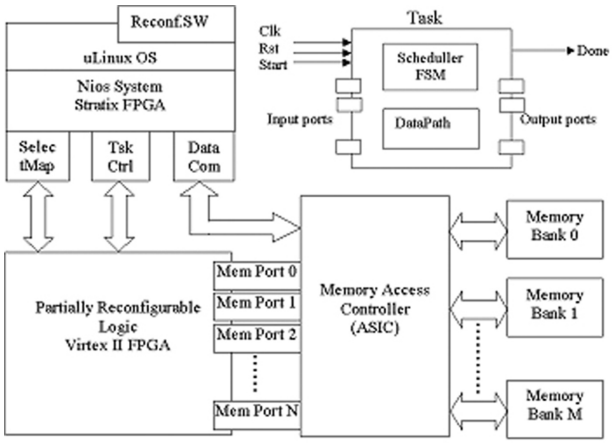


Figure 2. Platform architecture and task interface.

The tasks are implemented in a second FPGA, a Virtex-II Xilinx device [7]. This FPGA provides reconfigurable resources for tasks. Two ports allow the communication between the Virtex-II and Nios II System: The *SelectMap* allows the programming of contexts in the Virtex-II. A command sent to *TskCtrl* allows the resetting of all tasks in the FPGA and the starting of context execution. After context completion, a *Done* signal, in the *TskCtrl* port, is sent to Nios II System. This signal calls the *ReconfSW* for scheduling the new context. The Virtex-II FPGA is only used for task implementations, with a small additional glue logic.

Massive data applications needs high bandwidth memory systems [8]. In our architecture, the high I/O pin density of Virtex II FPGA is used for implementation of multi-access memory structure. For example the Virtex II XC2V2000[7] device has two million gates for task implementations and around 600 I/O pins, allowing the group of FPGA pins in at least 10 memory ports of 24M Address × 16Bits each. These ports are called *Mem Ports* (0 to *N*) in Figure 2 allowing *N+1* simultaneous read/write memory access in each context.

The communication between tasks are performed in the shared memory system and controlled by the *Memory Access Controller*. This is a high speed circuit that connects the *Mem Ports* of FPGA to several SRAM banks (*Bank* 0 to *Bank M* in Figure 2). One special port connected to host system (*DataCom*) allows the input and output of data in the memory array. This controller receives the access requisitions from FPGA tasks or host and responds to all accesses in only one system clock cycle (*Clk*). For this action a high speed connection is necessary between *controller* and *memory banks*. This connection is performed with a high speed clock called memory clock (*MClk*), which is several times bigger than the system clock (*Clk*). The typical internal structure of memory access controllers are discussed in [8]. Because the

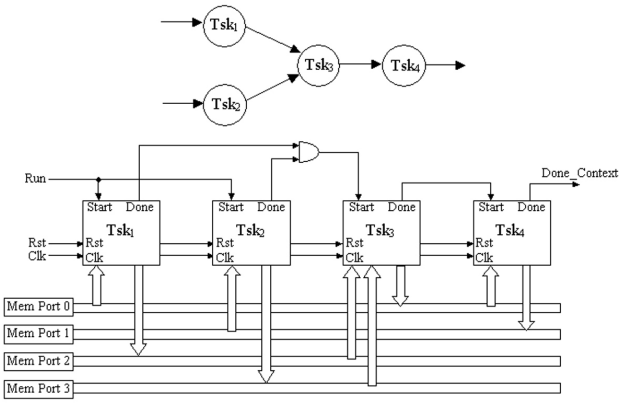


Figure 3. DFG application graph and FPGA task mapping.

memory access is performed during the task execution, the task communication time is included in the execution time of each task.

Figure 3 depicts a typical application DFG graph and the mapping of an application in one FPGA context. The DFG edges represent communication between tasks or data input/output and mapping in the memory system. The signals *Start* and *Done* of each task are combined in the glue logic according to dataflow in the DFG. These signals define the scheduling of tasks in each context. The *Run* signal from *TskCtrl* starts the context execution (Tsk_1 and Tsk_2) and *Done_Context* signals the context completion (end of Tsk_4 in the example in Figure 3) to Nios II host system.

The input and output ports of tasks are mapped to *Mem Port*s of FPGA. An arbiter, small glue logic, controls the access to the memories in case of simultaneous *Task Port* requests to same *Mem Port* in the FPGA. Each task port is composed of the following signals: *Address*, *Data*, *RA*, *WnR*, *W*, *nReq*. The *RA* is the request access signal. *WnR* defines the type of memory request: read or write. *W* is the wait signal, the bus arbitration uses this signal for ordering access to the *Mem Port buses.*

The *nReq* is generated when bus arbitration has pending requests. *nReq* blocks new task port requests while pending requests to remain in the *Mem Port* bus.

3. RELATED WORKS

Many works on reconfigurable computers have been described in literature. These works are in general classified into three classes: I-gate level partitioning; II-operation level partitioning and III-task level partitioning [5]. The gate level partitioning splits the application into subsets of gates. This is an adequate method for special devices with high reconfiguration speed (few clock cycles or micro-clock cycles) as multi-context FPGAs. Operation level partitioning is a very frequent approach where partitions are generated by grouping operand nodes from a DFG representation. In this approach, it is possible to consider the parallelism loop exploration and functional unit reuse in the same partitioning because of sequential execution of operands in the DFG. The classes I and II have two major problems: first these approaches are strongly related to the logic synthesis and high level synthesis with high granularity (gate and operation levels). Therefore, for high complexity applications the temporal partitioning procedure becomes very hard and there are several difficulties for hardware estimations during partitioning phase. The second problem is the difficulty for reuse of pre-synthesized tasks (from the library components). This reuse is fundamental for reduction of design effort and time-to-market in complex applications. Therefore, the task level partitioning is a way to task level reuse.

Several works in the task level partitioning and examples of image processing are found in literature: Vemuri et al. [13] proposes a task level partitioning method based on integer linear program (ILP)[14]. In this method the temporal partitioning and the selection of task implementation from library component is modeled by a set of binary variables $\{y_{tpm}\}$. The y_{tpm}=1 means that task t is mapping in the partitioning p with implementation m. If application has N_{Tsk} tasks, each task has N_{Impl} implementations and there is N_{Ctx} partitions, then the number of variables is $N_{Var} = N_{Tsk} * N_{Impl} * N_{Ctx}$. For example, if N_{Tsk} =17, N_{Impl} =5 and N_{Ctx} = 3 then N_{Var} = 255. The ILP problem is composed of several equations (in the variables y_{tpm}) that represent the temporal partitioning rules, resource limitations and application constraints. The problem of this approach is the high number of variables and equations that compose the ILP problem. Thus, for complex applications, the computational cost of ILP solution becomes very high.

Ouni et al., in [15], presents a method based on local search for temporal partitioning improvement. Ouni considers a library with several implementations for each task that constitutes the application. The partitions are generated by task grouping, from the application DFG. The smaller area (slower) implementation of each task is used in the initial partitioning. After partitioning, the execution time is calculated. The critical time partition is found and the worst task in terms of latency in the critical path has its implementation changed to the immediately faster implementation in the library. With the new implementation of the worst task, a new partitioning is generated. If the new one presents performance improvement then this procedure is repeated until that the there is no more increase in performance. The disadvantage of this approach is that after changing the task implementation a complete new partitioning procedure is performed. Therefore,

complex applications demand large partitioning times. Ouni reports partitioning times around 12 hours for applications with approximately 40 tasks.

Quinn et al. in [16] presents an automatic methods of component assignment for image processing pipeline in reconfigurable computers. Quinn suggests the use of hw/sw implementation library and three algorithms based on exhaustive search, local search and ILP. However, the application is limited to sequential tasks and each task has only one hardware and one software implementation. Another interesting work is presented in [17]. This work presents a codesign methodology for the UltraSONIC reconfigurable computer. This platform is designed for real-time video applications and a temporal partitioning and scheduling allows the mapping of tasks for hardware or software implementation. These tasks are executed under the control of a software task manager in the host processor, similar to our platform architecture. The limit in this approach is that only one implementation for each serial task in the FPGA is analyzed.

Finally the work presented in [18] proposes a temporal partitioning methodology that allows design space exploration generated by flexibility of task implementation. This methodology takes into account mathematical models for the *Area×Time* relationship for each task. The models and local search strategies are used to accelerate the choice of the best task implementation in the partitioning. This is a very interesting approach; however, the tasks in the applications are limited to sequential task flows and tasks with image processing algorithms based on windows of pixels.

4. DESIGN FLOW AND ALGORITHM DESCRIPTION

Our heuristic is based on hardware reuse paradigm (core library) and task level partitioning, where hardware synthesis and task design exploration are performed by hardware designers. Thus, the application experts can concentrate on choice of tasks as black-boxes, in a "lego design style". The proposed mapping of applications in the reconfigurable platform follows the design flow shown in Figure 4. The application is specified as a DFG graph where each node represents one task and the edges represent the data flow between tasks. The behavior of each task is a C or a SystemC code. Alternatively, the designer can use common application domain tasks stored in the component library.

For design space exploration of individual tasks, the use of high level synthesis tools is proposed. In this particular work, the Cynthesizer tool from Forte Design Company has been used [9].

This tool allows the translation of SystemC

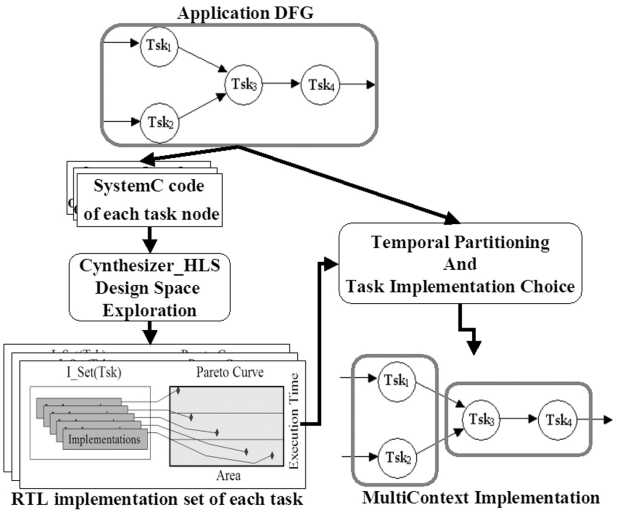


Figure 4. Proposed design flow.

behavior to several RTL implementations for a given target FPGA. The Cynthesizer allows the designer to explore the synthesis process by specification of several synthesis strategies from minimum area implementation to maximum speed implementation.

Cynthesizer allows a quick exploration of multiple algorithms and architectures to achieve a given set ***I_Set(Tsk)*** of execution times and area requirements for each task in the application. These implementation points are plotted in the pareto curves (Figure 1).

After that, the designer should evaluate and define the necessary number of temporal partitions for optimal time implementations. The designer should choose (from pareto curves) the best combination of task implementations that result in a optimal performance for each context partition that fit inside the FPGA area.

The increase in the number of partitions can be used to increase the available area, allowing the use of high speed task implementations. However, an excessive number of partitions results in performance degradation because of reconfiguration time overhead.

In this work an automatic *temporal partitioning and task implementations choice* algorithm is proposed. The Figure 5 shows the algorithm to find a good solution automatically, thus reducing the design effort.

A. The Temporal Partitioning Algorithm

The algorithm starts by generation of an initial partitioning. This partitioning groups task $Tsk_i \in A$ into successive π_a partitions in ASAP order of the tasks. If the area of π_a reaches the limit of the FPGA area A_{FPGA} then a new partition π_{a+1} is created for grouping the remaining tasks. In this initial partitioning each task Tsk_i is implemented with small area and small performance found in the pareto curve of imple-

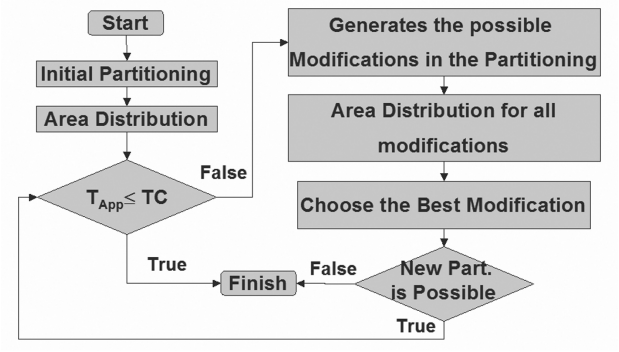


Figure 5. Automatic temporal partitioning algorithm.

mentation sets $I\_Set(Tsk_i)$, therefore a minimal number K of contexts is generated.

After initial partitioning, the *Area Distribution* procedure is applied. This procedure chooses the best implementations of tasks in each context (from pareto curve) that produces the most performance increase. This procedure improves the distribution of FPGA resources among tasks by changing used implementations. The main goal is the minimization of execution time of each context. After this procedure, the execution time of all application T_{App} (including the reconfiguration time) is compared with the time constraint TC. If TC is not reached the next algorithm steps are used for partitioning improvement.

The modifications in the partitioning can be provided in two ways, as shown in Figure 6: 1- move *tasks* between partitions; 2- create new empty partitions (*New CTX*) and move task to it.

Moving tasks can allow best use of FPGA area because this move creates an area in the original context and the moved task can use waste areas in the destination context for performance increase.

The creation of new contexts increase the effective area available for tasks and can improve the global performance. However, there is a supreme limit for the number of partitions (see Figure 6). This limit is given by the difference between the time constraint and minimal critical path time possible in the application *DFG*. This difference represents the maximum time available for context reconfiguration. Therefore, the limit is given by this difference divided by reconfiguration time T_{Rec}. Evidently, another limit is the number of tasks N_{Tsk} in the application.

In the algorithm of Figure 5 all possible modifications of current partitioning are generated and the *Area Distribution* procedure is applied to each new partitioning for choice of the best implementation of each task. After, the algorithm chooses the new partitioning, with the best execution time, from new current partitioning set.

An important observation is that the algorithm chooses the best time partitioning from new partitioning set, but this new partitioning can have poor performance in comparison to the previous one. This feature avoids possible local minimal during the partitioning search.

If a new partitioning reaches the time constraint TC, the temporal partitioning finishes with a good solution.

Two additional stop criteria for the algorithm are necessary: 1- The limit of number of contexts is reached or 2- A new temporal partitioning is not possible.

The temporal partitioning algorithm implements a local search method depicted in Figure 7.

Each dot in the search graph represents a temporal partitioning and the algorithm explores the best solutions in a direct way. A list of points visited is generated to avoid cyclical paths in the search, eliminating multiple visits to the same points. However, visited points list is emptied if the new current point in the search is generated by the addition of one new context (in the current partitioning). In this case, it is impossible that the algorithm reach a previously visited point. Therefore, in this case the list can be discarded. This reduces the memory used in the algorithm.

Despite the exponential complexity of total design space, the computational cost of our method is limited because of two aspects: 1- The granularity of task level partitioning used is coarse. This limits the number of objects in the partitioning; 2- The search strategies prunes the design space avoiding poor solutions. This reduces the size of the explored regions.

Our temporal partitioning includes the best task implementation choice called *Area Distribution* procedure. The efficiency of this procedure is fundamental for global efficiency of temporal partitioning. The next section describes this procedure in details.

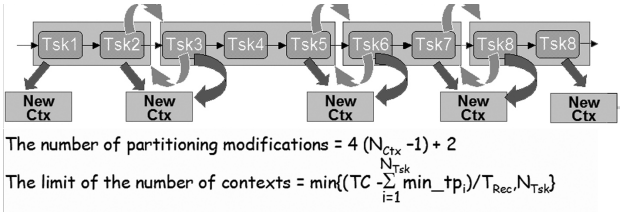


Figure 6. Modification of current partitioning.

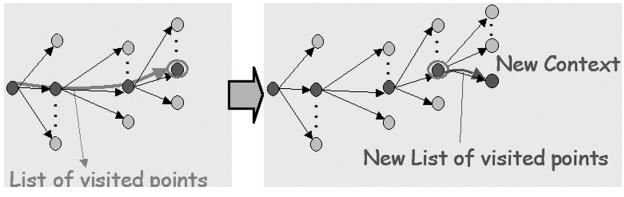


Figure 7. Local search given by temporal partitioning algorithm.

B. Area Distribution Procedure

The heart of the temporal partitioning proposed in this work is the *Area Distribution Procedure*. This procedure defines the best combination of task implementations for optimal execution time of each context. This procedure is responsible for the efficient utilization of FPGA resources and fundamental for generation of good partitioning. In Figure 8 the algorithm proposed for this procedure is depicted.

This procedure is applied to each context in each new partitioning generated in the algorithm of Figure 5. The algorithm starts with an initial solution for task implementations and the loop in Figure 8 refines this by changing the task implementation.

If the context has wasted resources ($\pi_a \leq A_{FPGA}$) then the algorithm chooses the best task for area increase that results in the best performance improvement.

However, if context area is larger than FPGA limit ($\pi_a > A_{FPGA}$), then the algorithm chooses the best task for implementation of area reduction that results in a reduced performance impact (smaller increase in the execution time). The loop is repeated until changes in the task implementations become impossible or area constraint reached. In each loop interaction, only one task implementation is modified and the task change results in the choice of next or previous task implementation in the pareto curve (Figure 1) that is relative to the current task implementation.

If *Area Distribution* procedure returns a context implementation with area superior to the FPGA area ($\pi_a > A_{FPGA}$), then the partitioning that contains this context is considered invalid. In the other case, the solution is considered a good context implementation. The computational cost of this procedure is given by the number $N_{Impl}(Tsk)$ of implementations for each task Tsk in the library. The superior limit LN_{Inter} for the number of loops for needed interactions of the *Distribution Algorithms* is given by:

$$LN_{Inter} = \sum_{\forall Tsk \in \pi_a} N_{Impl}(Tsk)$$

Therefore, the convergence of the algorithm to a final solution is guaranteed.

The final question in the description of the *Area Distribution Procedure* is the choice of an initial solution in the first step of the algorithm. In our approach, the procedure is executed and evaluated for three initial solutions. The best solution used for each context is the best one of the three solutions generated. The procedure is repeated for three initial solutions in order to avoid possible local minimal solutions. These initial solutions are the following:

1-*Large Area Initial Solution* (*LAIS*): Initial solution with the biggest area implementation for all tasks;

2-*Small Area Initial Solution* (*SAIS*): Initial solution with the smallest area implementation for all tasks;

3-*Optimized Initial Solution* (*OIS*): Initial solution generated by optimization method [14].

The *LAIS* and *SAIS* are very direct solutions given by the selection of extreme implementations in the pareto curve in Figure 1. However, the generation of *OIS* is more complex. The *OIS* is given by the following steps:

1-***Linearization of pareto curve***: A linear approximation of the implementation curve (pareto curve) is generated for each task in the application. An example of this approximation is depicted in Figure 9

The linear approximation is represented by the equation $f(T) = a + bT$ where $f(T)$ is the implementa-

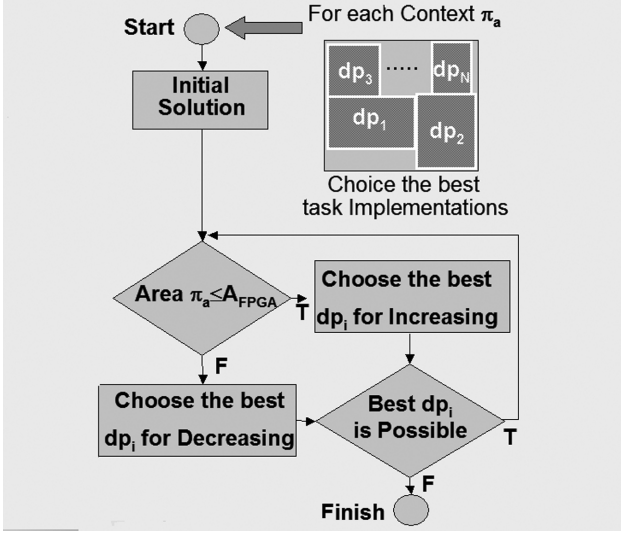


Figure 8. Area Distribution Algorithm.

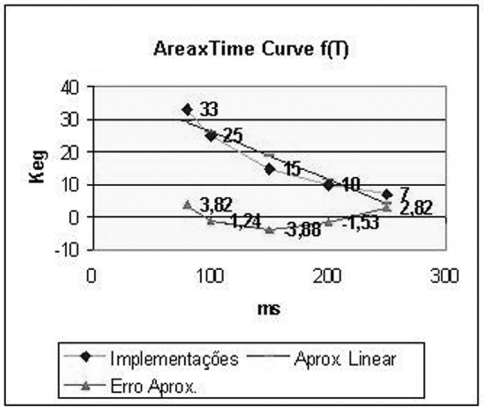


Figure 9. The linear approximation of pareto curve of task.

tion area in equivalent gates (*Keg*) and T is the execution time of implementation in milliseconds (*ms*). The coefficients a and b are given by the quadratic error minimization. For each task Tsk one equation $f_{Tsk}(T_{Tsk})$ is used for the characterization of tasks in the components library.

2-*The Optimization problem resolution*: For each task path $Path_i$ presents in the context π_a one linear optimization problem LP is constructed. This LP is composed of the equations:

Execution Time of Path:
$$ET_i = \sum_{\forall Tsk \in Path_i} T_{Tsk}$$

Critical Path Definition:
$$\forall\ Path_j \neq Path_i: \sum_{\forall Tsk \in Path_j} T_{Tsk} \leq ET_i$$

Area constraint:
$$\sum_{\forall Tsk \in \pi_a} f_{Tsk}(T_{Tsk}) \leq A_{FPGA}$$

The LP problem finds the values $T^{Tsk}{}_{Min}i$ of variables T_{Tsk} as real numbers that minimize ET_i with restrictions given by the above inequalities. This problem is applied for each path $Path_i$ in the context π_a and the minimal time solution for the context is given by:

$Solution(\pi_a) = \{T^{Tsk}{}_{Min}i \mid \forall Tsk \in \pi_a\}$ where $ET_i = min\ \{ET_j \mid \forall Path_j \in \pi_a\}$

The solution above represents the execution times $T^{Tsk}{}_{Min}i$ for each task Tsk that minimizes the execution time of context π_a. However, because the LP is resolved in the continuous domain (Real field), the times $T^{Tsk}{}_{Min}i$ do not correspond to the implementations present in the pareto curves. Therefore, an adjustment is necessary for the choice of implementations in the pareto curve with execution time near to the times of $\{T^{Tsk}{}_{Min}i\}$. The choice of implementation $Impl(Tsk)$ for task Tsk in the pareto curve is given by the following rule:

Choice $Impl(Tsk)$ *such that the execution time is the nearest possible to time* $T^{Tsk}{}_{Min}i$ *(given by the* LP *problem). If two implementations are equidistant of* $T^{Tsk}{}_{Min}i$, *then the* $Impl(Tsk)$ *with smaller execution time implementation should be chosen*∇

The utility of *OIS* as initial solution, as described above, is that this solution can be near the globally optimal implementation points. Therefore, the probability of the *Area Distribution Procedure* reaches local minimal and poor solution is reduced. In addition, the number of necessary interactions in the algorithm is minimized.

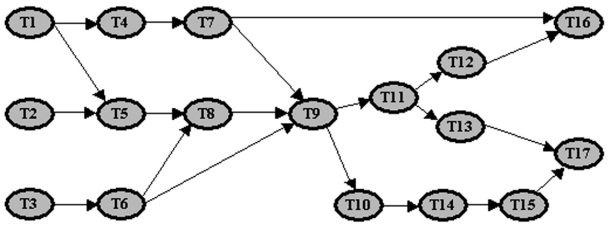


Figure 10. The application graph for experiments.

In this work, the solution of the LP is generated by optimization toolbox of the MatLab [19]. The *Temporal Partitioning* algorithm with *Area Distribution Procedure* is implemented as a Maltlab program and executed in the Matlab environment. However, the speed of partitioning can be increased by its translation to C code.

In the next section, experimental results using synthetic application graphs area presented. The results demonstrate the quality of our approach.

5. EXPERIMENTAL RESULTS

For the validation of our approach the algorithms depicted in Section 4 are applied to synthetics task graph shown in the Figure 10.

Each task (*T1* to *T17*) has five possible implementations in each *Area_Time_Set* (*ATS*) as presented in Table 1 below (see definition of *ATS_i* sets in last line of table). Each set of values in the *ATS_i* is used in an experimental evaluation of our approach.

These experiments use three sub-graphs *DFG_1, DFG_2* and *DFG_3* (see Table 2) from graph in Figure 10. Therefore, 9 experiments had been executed as indicated in Table II below.

The *DFG* of Figure 10 and values in the Table 1 are generated by hand and they represent typical topologies of real applications and typical implementation of tasks. The sets *ATS_i, i=1,2,3* present differences of dispersion of area and time values between the tasks.

These differences allow the representation of several possibilities in the real world cases. In the next sections the results of *Area Distribution* algorithm and one case of *Temporal Partitioning* is analyzed in detail

A. Area Distribution Results

The *Area Distribution* Algorithm described in the Section *4B* was applied in the experiments of Table 2. In each case the *DFG* was mapped for only one context. Several reconfigurable areas for the context have been considered in the range of minimal area necessary for implementation of *DFG* (with smaller

Table I. Implementations of tasks used in the our experiments (synthetic example)

	Implementations (Area in Kequivalent gates and Time in miliseconds)									
Tasks:	**Imp 1**		**Imp 2**		**Imp 3**		**Imp 4**		**Imp 5**	
	Area	**Time**	**Area**	**Time**	**Area**	**Time**	**Area**	**Time**	**Area**	**Time**
T1:	5/5/5	20/20/20	9/9/9	14/14/14	12/12/12	10/10/10	15/15/15	8/8/8	17/17/17	7/7/7
T2:	7/7/7	250/25/25	10/10/10	200/20/20	15/15/15	150/15/15	25/25/25	100/10/10	33/33/33	80/8/8
T3:	3/3/3	125/125/125	7/7/7	100/100/100	12/12/12	70/70/70	17/17/17	50/50/50	20/20/20	45/45/45
T4:	30/30/30	450/45/45	40/40/40	300/30/30	70/70/70	180/18/18	100/100/100	140/14/14	125/125/125	120/12/12
T5:	18/18/18	230/23/23	25/25/25	200/20/20	50/50/50	160/16/16	90/90/90	150/15/15	110/110/110	80/8/8
T6:	27/27/27	270/27/27	50/50/50	200/20/20	80/80/80	130/13/13	100/100/100	105/10/10	120/120/120	96/9/9
T7:	80/80/8	260/26/26	160/160/16	130/13/13	250/250/25	90/9/9	330/330/33	60/6/6	400/400/40	40/4/4
T8:	56/56/5	105/10/10	120/120/12	90/9/9	200/200/20	70/7/7	230/230/23	50/5/5	300/300/30	45/4/4
T9:	150/150/15	500/50/50	300/300/30	260/26/26	400/400/40	190/19/19	500/500/50	130/13/13	670/670/67	90/9/9
T10:	80/80/8	250/25/25	100/100/10	200/20/20	120/120/12	180/18/18	200/200/20	105/10/10	250/250/25	90/9/9
T11:	20/20/20	30/30/30	40/40/40	16/16/16	65/65/65	10/10/10	90/90/90	8/8/8	105/105/105	7/7/7
T12:	500/500/50	100/10/10	600/600/60	80/8/8	730/730/73	73/7/7	840/840/84	62/6/6	900/900/90	51/5/5
T13:	450/450/45	60/60/60	700/700/70	50/50/50	800/800/80	45/45/45	850/850/85	40/40/40	960/960/96	36/36/36
T14;	5/5/5	15/15/15	20/20/20	8/8/8	35/35/35	5/5/5	57/57/57	4/4/4	75/75/75	3/3/3
T15:	37/37/37	10/10/10	49/49/49	8/8/8	60/60/60	6/6/6	65/65/65	5/5/5	70/70/70	4/4/4
T16:	100/100/10	8/8/8	200/200/20	4/4/4	250/250/25	3/3/3	300/300/30	2/2/2	370/370/37	1/1/1
T17:	70/70/7	24/24/24	140/140/14	12/12/12	180/180/18	10/10/10	200/200/20	9/9/9	250/250/25	7/7/7

ATS Definition: value_1/Value_2/Value_3 in each cell of table means that value_i is used in the Area_Time_Set number i (ATS_i) with i=1,2,3.

Table II. Set of executed experiments

Used Subgraphs:
DFG_1 = {T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13,T14,T15,T16,T17}
DFG_2 = {T1,T2,T3,T4,T5,T6,T7,T8,T9}
DFG_3 = {T9,T10,T11,T12,T13,T14,T15,T16,T17}

Experiment:	**Used SubGraph**:	**Used ATS**:
Exp_1:	DFG_1	ATS_1
Exp_2:	DFG_1	ATS_2
Exp_3:	DFG_1	ATS_3
Exp_4:	DFG_2	ATS_1
Exp_5:	DFG_2	ATS_2
Exp_6:	DFG_2	ATS_3
Exp_7:	DFG_3	ATS_1
Exp_8:	DFG_3	ATS_2
Exp_9:	DFG_3	ATS_3

area of each task given by *Impl 1* column in Table 1) to the maximum area necessary for implementation of *DFG* (with greater area of each task in the *Impl 5* column of Table 1). The results shown in the Table 3 are compared with results generated by exhaustive search for best solution in the complete design space for subgraphs *DFG_2* and *DFG_3* (*Exp_4* to *Exp_9* in Table 2).

In the exhaustive search all possible combinations of task implementations are analyzed and the globally optimal solution with the best execution time (*E* in Table 3) is selected. Because of very large size of design space for the experiments *Exp_1, Exp_2, Exp_3*, in order of 5^{17} = 7.63 x 10^{11} points, the exhaustive search was not performed. For experiments *Exp_4* to *Exp_9* the size of design space is 5^9 = 1953125 points. In this case exhaustive search C program consummates ~11 seconds in the platform Celeron M – 1.3GHz -256MBytesRAM – WindowsXP-2002.

The results for experiments *Exp_4* to *Exp_9* demonstrate that our approach generates solutions (choice of implementations of tasks) very near globally optimal solutions with average error $Er \leq$ 3.5% and maximum error around 13% in Table 3.

Figure 11 shows the number of loop interactions for experiment *Exp_1* in function of FPGA Area.

The number of interactions is very constant, independently of FPGA size. The number of loop interactions of algorithm (in Figure 8) necessary for solution generations is smaller than 100 for all experiments. This is a very fast convergence compared with large size of design spaces. Therefore, our approach for task implementation choice combines high precision and high speed in the solution generation. As described in Section *4B* the *Area Distribution Procedure* is executed for three initial solution *LAIS*, *SAIS* and *OIS*. The final solution is the best of three possible solutions. Figure 12 shows the generated solutions for each initial solution.

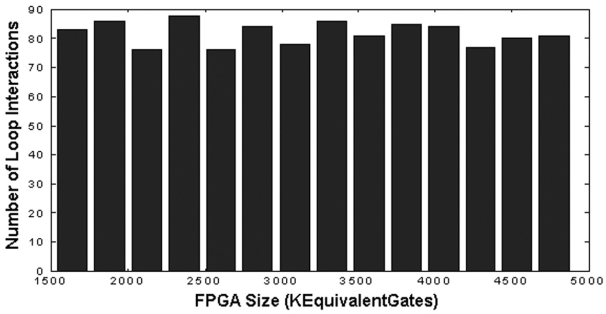


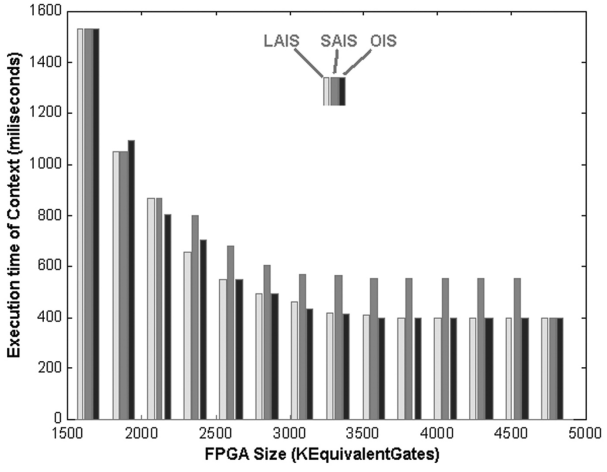
Figure 11. Number of loop interactions.

Table III. Experimental results of Area Distribution algorithm for experiments in the Table II

Used Symbols: T= Execution time results (milisecond); A= Several sizes of FPGA area (Kequivalente gates) for context implementation; E = represents the best execution time obtained by exhaustive search; Er= (T-E)/E represents the relative error of our algorithm, m= Maximum values of error Er; a= Average values of Er

Exp_1	A-1638	A-1879	A-2121	A-2362	A-2603	A-2845	A-3086	A-3327	A-3569	A-3810	A-4051	A-4292	A-4534	A-4775	Er:
	T-1529	T-1050	T-804	T-704	T-549	T-494	T-431	T-413	T-399	T-399	T-399	T-399	T-399	T-399	-
	E- -	E- -	E- -	E- -	E- -	E- -	E- -	E- -	E- -	E- -	E- -	E- -	E- -	E- -	
Exp_2	A-1638	A-1879	A-2121	A-2362	A-2603	A-2845	A-3086	A-3327	A-3569	A-3810	A-4051	A-4292	A-4534	A-4775	Er:
	T-326	T-200	T-169	T-159	T-143	T-134	T-126	T-120	T-117	T-117	T-117	T-117	T-117	T-117	-
	E--	E--	E--	E--	E--	E--	E--	E--	E--	E--	E--	E--	E--	E- -	
Exp_3	A-300	A-360	A-421	A-481	A-542	A-602	A-662	A-723	A-783	A-844	A-904	A-964	A-1025	A-1085	Er:
	T-326	T-201	T-168	T-148	T-132	T-124	T-118	T-117	T-117	T-117	T-117	T-117	T-117	T-117	-
	E- -	E--	E--	E--	E--	E--	E--	E--	E--	E--	E--	E--	E--	E- -	
Exp_4	A-376	A-485	A-594	A-704	A-813	A-922	A-1031	A-1140	A-1249	A-1358	A-1468	A-1577	A-1686	A-1795	Er:
	T-1230	T-925	T-715	T-680	T-577	T-575	T-465	T-450	T-375	T-360	T-340	T-320	T-300	T-295	m12.92%
	E-1230	E-925	E-715	E-638	E-575	E-515	E-447	E-395	E-370	E-347	E-340	E-307	E-297	E-295	a3.35%
Exp_5	A-376	A-485	A-594	A-704	A-813	A-922	A-1031	A-1140	A-1249	A-1358	A-1468	A-1577	A-1686	A-1795	Er:
	T-212	T-116	T-101	T-90	T-83	T-78	T-73	T-72	T-68	T-68	T-67	T-67	T-67	T-67	m1.5%
	E-212	E-116	E-101	E-90	E-83	E-77	E-73	E-71	E-68	E-67	E-67	E-67	E-67	E-67	a0.2%
Exp_6	A-118	A-152	A-186	A-220	A-255	A-289	A-323	A-357	A-391	A-425	A-460	A-494	A-528	A-562	Er:
	T-212	T-113	T-95	T-85	T-80	T-73	T-68	T-67	T-67	T-67	T-67	T-67	T-67	T-67	m2.81%
	E-212	E-113	E-94	E-85	E-78	E-71	E-68	E-67	E-67	E-67	E-67	E-67	E-67	E-67	a0.46%
Exp_7	A-1412	A-1584	A-1756	A-1929	A-2100	A-2273	A-2445	A-2617	A-2789	A-2961	A-3133	A-3306	A-3478	A-3650	Er:
	T-799	T-509	T-398	T-272	T-232	T-208	T-200	T-194	T-194	T-194	T-194	T-194	T-194	T-194	m1.52%
	E-799	E-509	E-398	E-272	E-231	E-207	E-197	E-194	E-194	E-194	E-194	E-194	E-194	E-194	a0.17%
Exp_8	A-1412	A-1584	A-1756	A-1929	A-2101	A-2273	A-2445	A-2617	A-2789	A-2961	A-3134	A-3306	A-3478	A-3650	Er:
	T-164	T-126	T-104	T-92	T-87	T-76	T-67	T-63	T-59	T-59	T-59	T-59	T-59	T-59	m2.63%
	E-164	E-126	E-104	E-92	E-87	E-76	E-67	E-63	E-59	E-59	E-59	E-59	E-59	E-59	a0.55%
Exp_9	A-197	A-227	A-257	A-288	A-318	A-348	A-378	A-409	A-439	A-469	A-499	A-530	A-560	A-590	Er:
	T-164	T-125	T-104	T-91	T-74	T-66	T-62	T-60	T-59	T-59	T-59	T-59	T-59	T-59	m1.69%
	E-164	E-125	E-104	E-91	E-74	E-66	E-62	E-59	E-59	E-59	E-59	E-59	E-59	E-59	a0.12%

As shown in Figure 12 the best solution can be generated by any of the three initial solutions. The efficiency of each initial solution for generation of best final solution is given in Table 4.



Figue 12. Generated solution in function of initial solution.

Table IV. Efficiency of Initial Solutions for Area Distribution

Initial Solution:	**Generator of the best solution:**		**Only generator of the best solution:**	
	Exp1to3	Exp4to9	Exp1to3	Exp 4to9
LAIS:	69.05%	81.25%	23.52%	22.73%
SAIS:	33.33%	58.33%	5.88%	4.55%
OIS:	85.71%	83.33%	70.59%	72.73%

The first two column values in Table 4 show the percentage of cases where each initial solution results in the best final execution time. In this case other initial solutions that results in the same execution time are possible. The last two columns represent the percentage of cases where each initial solution is the only initial solution that results in the best execution time. These results demonstrate that the most efficient initial solution is the *OIS* (*Optimized Initial Solution*). However, the initial solutions *LAIS* and *SAIS* are very important because in 27.28% of the cases the best solutions are given by these initial solutions. The combination of this *Area Distribution* procedure for these three initial solutions generates very good results: in 65.15% of the cases the final result is precisely the global minimal time of execution time for each context and the remaining 34.85% of cases result in small errors as shown in Table 3.

The solution of ***LP*** problem for initial solution *OIS* is generated by the MatLab optimization toolbox in some seconds and it represents small computation time overhead in the *Area Distribution* procedure.

The results presented in this section demonstrate the efficiency of our approach for choice of best task implementations in each context of temporal partitioning. In the next section, the use of these results in the temporal partitioning procedure is shown.

B. An Example of Temporal Partitioning

The temporal partitioning procedure was applied to the Fig. 10 graph with *Area_Time_Set* and FPGA area of 820 Kequivalentes gates. The typical reconfiguration time *TRec* = 16.4msec is considered.

The initial partitioning results in three contexts π_1, π_2 and π_3 as follow: π_1={T1,T2,T3,T4,T5,T6,T7,T8,T9, T10, T11,T14,T15}, π_2= {T13,T17} and π_3= {T12,T16}.

This results in the total reconfiguration time of 49.20ms and the total execution time of application is 1746.20ms. The application of *Area Distribution* procedure increases the performance of this initial partitioning and the new execution time resultant of task choice is 1157.20ms.

The main loop of temporal partitioning in Fig. 5 can improve this partitioning by changes in the partition mapping of tasks. As example, some interactions of the loop result in the following new partitioning:
π_1= {T1,T2,T3,T4,T5,T6,T7,T8,T9,T10},
π_2= {T11,T13,T14,T15,T17} and π_3= {T12,T16}

This new partitioning, after the application of *Area Distribution Procedure*, results in the execution time of 1027.20ms. These results represent a performance increase of (34%) and (11%) respectively.

Finally, let's consider that the memory resources are sufficient for context implementations. In the next version of partitioning approach, a task scheduler mechanism will be introduced to consider the memory constraints.

6. CONCLUSIONS AND FUTURE WORKS

A reconfigurable computer architecture and design flow have been presented for task implementation exploration in massive data processing applications. This design flow allows the reuse of tasks implementations from a components library. This reuse allows the reduction of design effort. The methodology is based on an application model that can be applied in applications composed of real world tasks as FFT, DCT, Filters, Edge Detector, numeric intensive simulations, etc and integrate high level synthesis tools as, for example, Forte – Cynthesizer.

An efficient algorithm for design space exploration of task implementations inside temporal partitioning is presented and experiments have been shown. The results demonstrate the viability and efficiency of our approach.

Although the reconfiguration time can represent one bottleneck, high speed implementation of tasks in the massive data applications (possible with effective area increase) can compensate for these reconfiguration times and decrease the application execution time.

As future works, we intend to complete integrations of platform and design flow environment, the use of more complex and real applications and include several application domains. The inclusion of memory access scheduling in the *DFG* application for memory constraint modeling is under development.

ACKNOWLEDGEMENTS

This research is partially supported by the Brazil agencies: CNPq and FACEPE.

REFERENCES

[1] Bruce A. Draper, Ross Beveridge, A.P. Willem Böhm, Charles Ross, and Monica Chawathe, "Accelerated image processing on fpgas", IEEE Transactions on Image Processing, no. 12, 2003.
[2] Compton, Katherine; Hauck ,Scott; "Reconfigurable Computing: A Survey of Systems and Software"; ACM Computing Surveys, Vol 34, No 2, pp. 171-210, June 2002.
[3] Rhett D. Hudson, David I. Lehn, Peter M. Athanas ,"A Run Time Reconfigurable Engine for Image Interpolation", Bradley Department of Electrical and Computer Engineering Virginia Tech, Blacksburg, Virginia.
[4] Camel Tanougast, Yves Berviller Serge Weber, Philippe Brunet, "A Partitioning Methodology that Optimises the Area on Reconfigurable Real-Time Embedded Systems", EURASIP Journal on Applied Signal Processing, pp 494-501, Hindawi Publishing Corporation, 2003.
[5] Christian Plessl, Marco Platzner, "Virtualization of Hardware – Introduction and Survey", Computer Engineering & Networks Lab- Swiss Federal Institute of Technology(ETH) Zurich, Switzerland-2004.
[6] www.altera.com.
[7] www.xilinx.com.
[8] J.Corbal, R.Espasa, and M.Valero, "Command vector memory systems: High performance at low cost", In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 1998.
[9] Forte Design System; "Cynthesizer User's Guide for Cynthesizer 2.4.0"; www.forteDS.com; March 14, 2005.
[10] Torresen, Jim; Bakker, Jorgen W.; Sekanina, Lukas; "Efficient Image Filtering and Information Reduction in Reconfigurable Logic"; Proceedings of Norchip04, 2004
[11] Clouard, Regis; et al.; "Borg: A Knowledge-Based System for Automatic Generation of Image Processing Programs", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 21, NO. 2, February 1999.
[12] Gries, Matthias; "Methods for Evaluating and Covering the Design Space during Early Design Development"; Technical Memorandum UCB/ERL M03/32, CAD-Group, Electronics Research Laboratory August 12, 2003.
[13] Vemuri, Ranga; Kaul, Meenakshi; "Temporal Partitioning Combined with Space Exploration for Latency Minimization of Run-Time Reconfigured Design"; DATE 1999.
[14] Pierre, Donald A.; "Optimization Theory with Applications"; Dover Publications ,Inc.,New York 1986.
[15] Ouni, B.; Mtibaa, A.; Abid, M.; "Synthesis and Time Partitioning for Reconfigurable Systems"; Design Automation for Embedded System, 9, pp. 177-191, Springer – 2005.
[16] Quinn, Hearther; et.al; "Runtime Assignment of Reconfigurable Hardware Components for Image Processing Pipelines"; 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines FCCM-2003.
[17] Wiangtong, Theerayod; et. al.; "Hardware/Software Codesign – A systematic approach targeting data-intensive applications"; IEEE Signal Processing Magazine, May 2005.
[18] Nascimento, P. S. B.; Lima, M. E. ; "Temporal Partitioning for Image Processing Based in Reconfigurable Architectures"; DATE06 – Design Automation and Test in Europe; 6-10 March, Munich Germany 2006.
[19] http://www.mathworks.com.