Dynamic and On-Line Design Space Exploration for Reconfigurable Architectures

Fakhreddine Ghaffari[1,2], Michel Auguin[1], Mohamed Abid[2], and Maher Ben Jemaa[2]

[1] I3S, University of Nice Sophia Antipolis, CNRS, les Algorithmes bat. Euclide, 2000, route des Lucioles BP 121, 06903 Sophia-Antipolis
ghaffari, auguin@i3s.unice.fr
http://www.i3s.unice.fr/I3S/FR/

[2] Université de Sfax TUNISIA, Computer, Electronics and Smart engineering systems design (CES),
ENIS, BPW 3038 Sfax, Tunisie
maher.benjemaa, mohamed.abid@enis.rnu.tn

Abstract. The implementation of complex embedded applications requires a mix of processor cores and HW accelerators on a single chip. When designing such complex and heterogeneous System on Chip (SoCs), the HW/SW partitioning needs to be made prior to refining the system description. Traditional system partitioning is generally done at the early stage of system architecture, by defining the tasks to be implemented on the embedded processor(s), and the tasks to be implemented on the hardware. We describe here a new approach of On-line Partitioning Algorithm (OPA) which consists of adapting dynamically the architecture to the processing requirements. A scheduling heuristic is associated to this partitioning approach. We consider soft real time data flow graph oriented applications for which the execution time is dependent on the content of input data. The target architecture is composed of a generic processor connected to a dynamically reconfigurable hardware accelerator. The dynamic reconfiguration allows the self adaptation of the architecture which avoids redesigning a new architecture according to variation of characteristics of applications algorithms. We compare our method with an Off-line static HW/SW partitioning approach. We present results of the OPA on an image processing application. Our experiments included simulation results with SystemC for on-line scheduling and partitioning approaches. An ILP solver is used to compare the experiment results with an off-line static HW/SW partitioning approach.

1 Introduction

Modern complex embedded real-time systems require significant computational power while guaranteeing latency and timing performance. Guaranteeing the performance of a multi-tasked system often requires a far more powerful processor if we minimize embedded resources as well. Hybrid hardware-software

P. Stenström (Ed.): Transactions on HiPEAC I, LNCS 4050, pp. 179–193, 2007.

systems have a number of advantages over traditional microprocessor-based software systems or custom ASIC hardware solutions. The implementation of control-flow algorithms is difficult in hardware, while algorithms involving significant parallel arithmetic operations may be difficult to realize in a microprocessor-based software solution. Hybrid hardware software systems allow the parallelism to be exploited in hardware, while leaving control of the overall system in software. This may result in superior real-time performance, as argued in [1]. The aim is to adjust the computational power of current multimedia portable devices (such as embedded camera) while keeping their flexibility. Flexibility is required because different algorithms will run on the device, with different architecture requirements. Moreover, it enables upgrading and downloading of new applications. Reconfigurable hardware meets these two requirements and is therefore a valid solution for this problem. Hardware/Software partitioning is the process of dividing an application among software (running on microprocessors) and hardware units. Extensive research has shown that Hardware/ Software partitioning can result in overall software speedups [2, 3, and 4] as well as reducing system energy [14, 5 and 6]. Many applications, in particular in image processing (e.g. an intelligent embedded camera), have dependent data execution times according to the nature of the input to be processed. This kind of applications is often stressed by real time constraints, which demand adaptive computation capabilities. To partition data-dependent tasks on a heterogeneous architecture, new design approaches are necessary. Particularly for applications with soft real time constraints, we aim to minimize the embedded resources so as to avoid an architecture composed of the resources associated with the worst case execution times (WCET) of the functionalities. There is little work in the literature, which addresses this problem. The approach presented in [13] is based on an on-line HW/SW migration of tasks according to their execution times. This migration process is only applied locally to the most time consuming loop of the application program. The choice of dynamic re-allocation of the tasks presented in [13] is manual. The primary contribution of our work, though, is an extensive examination of the number of hardware resources savings as well as possible speedups through on line hardware/software partitioning. We have simulated our approach with a SystemC platform having a microprocessor coupled with a configurable logic on a real time image processing applications. The paper is organized as follows. Section 2 presents related works on HW/SW partitioning. Section 3 introduces the advances in dynamic reconfigurable systems. Section 4 presents the on-line partitioning algorithm. Section 5 shows the experimental results and finally we conclude in section 6.

2 Related Work

Recent works have introduced dynamic hardware/software partitioning [13]. During execution of an application, an on-chip profiling method detects critical regions of code for hardware implementation. An on-chip tool transparently

re-implements those regions on FPGA (Field Programmable Gate Array). Researchers have explored other dynamic optimization approaches. For example, Dynamo performs dynamic software optimizations on the most frequently executed regions of code [1]. The ProfileMe approach [15] specialize subroutines for common inputs and determines by runtime profiling which configuration to call for the best performance. In [16] the performance is improved by remapping frequently executed regions of code to non-interfering cache locations. Value profiling [17] determines runtime invariant variables for constant propagation and code specialization for optimized performance, or even for reduced energy. The appearance of single-chip platforms incorporating a microprocessor and FPGA in a single chip [7, 8 and 9] has recently made the hardware/software partitioning problem even more attractive. Such platforms yield more efficient communication between the microprocessor and FPGA than using two chip designs, resulting in improved performance and reduced power. By considering the FPGA as a fast extension of the microprocessor, a designer can move critical software regions from the microprocessor onto the FPGA hardware, so as to improve performance whereas the physical architecture is unchanged.

3 Advances in Dynamic Reconfigurable Systems

In the literature, some reconfigurables architectures have been proposed to implement a dynamic partitioning approach. The architecture proposed in [11] is formed by a reconfigurable logic targeted by the dynamic HW/SW partitioning. During the design of this architecture, the main goal was to minimize the runtime of the on line reconfigurable placement and routing. In [12] an approach of tasks re-allocation between hardware and software units is presented. It details the communication after switching the implementation of a task from a unit to another. An embedded operating system has been used to manage the different communications, context saving, the placement/routing and memory management.

The target architecture in our approach is composed of a processor connected to a Reconfigurable Computing Unit (RCU) through an intelligent interface conceived by the CEA[1] and called ICURE [10]. This architecture is schematized in figure 1. The considered embedded processor can be of any type provided that it allows an efficient coupling with the RCU. The RCU reconfiguration is achieved by a dedicated unit situated in the ICURE interface. The reconfiguration data are stored in a structure called Contexts. This latter contains the bit-stream, the routing information and the object code necessary to the reconfiguration and execution of the hardwired mapped tasks. The CPU has access to ICURE functionalities through API (Application Programming Interface) allowing a transparent utilization of the reconfigurable resources.

[1] CEA: Commissariat lEnergie Atomique (Research Comitee on atomic energy).

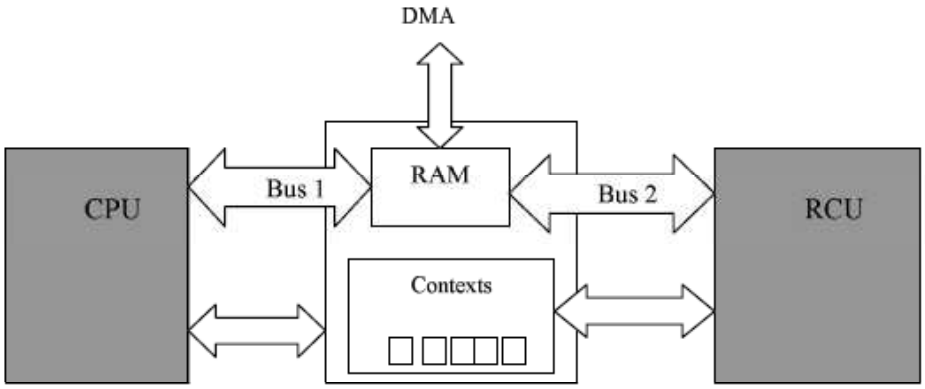

Fig. 1. The generic architecture

4 The On Line Partitioning Algorithm (OPA)

4.1 General Description

On-line HW/SW Partitioning has several important advantages over off-line approaches. OPA allows for a system to be optimized based on runtime behaviors and values, which may be hard to determine using off-line methods or costly simulations, and which also may change according to the environment with which the system interacts. Furthermore, on-line optimizations require no designer intervention and are applied transparently during runtime. In applications such as image processing, the execution of data dependent tasks consumes time accordingly to the characteristics of the incoming image. This time variation results from one or more Correlation Parameters (CP) in every processed image. An example of CP is the number of white pixels in an image or the number of moving objects. The goal of the On-line Partitioning Algorithm (OPA) is to dynamically allocate resources of the architecture to the tasks and schedule them such that the constraints are not violated. The OPA considers periodically the set of available embedded resources in the architecture.

The normal execution flow of the system on a sequence of input data In-1, In, In+1,... is illustrated in figure 2. During period In-1 the execution time of each task and of the whole application (Te) for the next period is estimated. While no violation of time constraints is estimated (Te ¡ Tmax), the current mapping and scheduling will be saved for the next period. Otherwise, a new partitioning/scheduling of the tasks is computed as depicted in period In in figure 2. The new mapping must deal with the available resources in the reconfigurable hardware unit and must provide a solution with an execution time less than the time constraint.

Partitioning is needed when violation of constraints are predicted. The basic idea consists to consider the current mapping and to carry out migrations of tasks between resources to satisfy the overall constraints (see period In+1 in

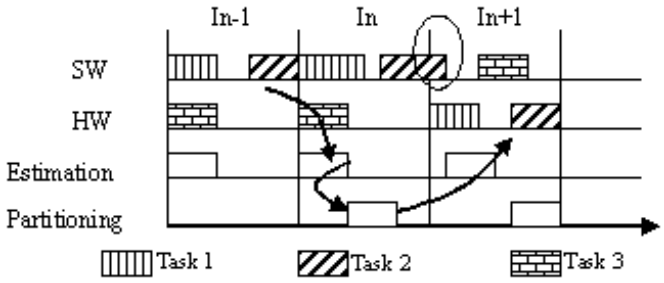

Fig. 2. Adaptation of partitioning to the processing need

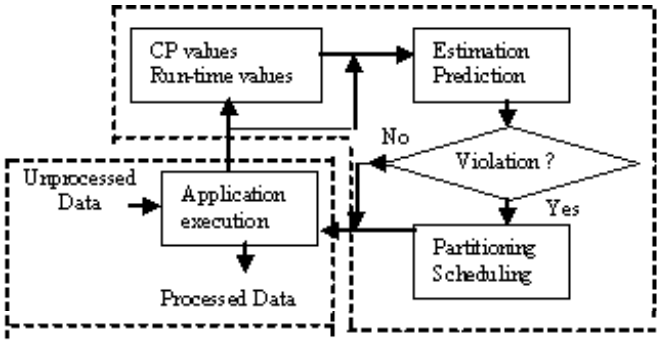

Fig. 3. Dynamic reconfigurable platform

figure 2). Assume that time constraint violation is predicted at period In and the task Ti is assigned to the processor. The current available area in the HW unit is Sn. A migration of Ti to the HW at the next period leads to a benefit of ?i on its execution time and the available area will be reduced to Sn+1 ¡ Sn. The On-line partitioning and scheduling are time consuming operations. Therefore the challenge is to develop fast efficient algorithms that can interact with the application functionalities and respect the time constraints of the application. Data flow-based applications are good candidates for this new approach. For example, while the system processes the image n, OPA performs the partitioning and the schedule of the estimations for image n+1. The figure 3 illustrates the functionalities embedded in an OPA system composed of two parts: the resources dedicated to the application itself and the OPA loop. The former is data flow oriented such as video, image, sound or any iterative data processing application with data dependent execution time. The application has its own internal data communications between dependant tasks, but only the exchanged data with the OPA loop subsystem is shown. The tasks of the application are executed according to the order given by the Scheduler. The execution time of each task and the associated CP value are collected in a database. At the end of each period, the OPA estimates the run time of each application task for the next

period based on the measures in the database. If the estimated schedule respects the timing constraints, the OPA loop will stop and the system executes the next period. In the other case the algorithm runs the Partitioning function and tries to find a partition which respects the constraints. This global view of the algorithm leads to questions about its integration in the whole system, and how complex will be its implementation.

4.2 System Model

To model the application we consider a Data Flow Graph (DFG), where nodes are processing functions (tasks) and edges describe communication between tasks. The size of the DFG depends on the number of tasks and edges that have a great effect on the execution times of the partitioning and scheduling operations. A low processing granularity of the DFG makes the system easy to be predictable because data dependent task execution time can depend on a single CP. In opposite a low granularity DFG needs more computing to complete the algorithm loop on a DFG of great size, leading to overstep the image processing period. Each node of the DFG represents a specific task in the application, and it can run either on software (processor) or on hardware (RCU). The aim of using reconfigurable hardware is simply obtaining the same flexibility while executing an application task on hardware as on software. By migrating a task from software to hardware and conversely, we make a significant change on the execution time. Any migration of task to the software will free the reconfigurable area, but will consequently increase the task run time. In opposite, any migration of task to hardware will have the reverse effects. Thus, it is important to have the suitable choices of task migration, and the choices are made in the lap of time between two periods. If the video processing constraint is 40 ms for each image, then the application execution time and the OPA execution time should be under that limit. Another difficulty lies on having low resources OPA cost compared with the cost of the resources needed to execute the application tasks; else, it will not be beneficial to overload the system with the OPA approach. To resume, the On-line Partitioning Algorithm tries to update the partition of an application when it is running. The algorithm must make choices between all tasks migrations and schedule them. Its cost (run time and resources) must be small compared with the overall cost of the system. The following sections will explain clearly each OPA function in the figure 3.

4.3 Prediction Algorithm

The efficiency of the system depends on the estimation accuracy. With the estimated values, for the next period, the OPA decides if the partition has to be changed. To estimate the future execution time of a task (TEXE), we have considered a simple interpolation equation. The approach of estimating with an interpolated function considers tasks whose execution time depends only on one Correlation Parameter. The estimation of such task could be done by a polynomial equation, which is found in an off-line analysis or profiling of the task code.

The CP is the only unknown variable in the equation, consequently only the estimation of the CP is needed to know the estimated execution time. For each task, the equation could be different according to the implementation. Estimating the execution time with a polynomial equation is rather simple and efficient but an off-line analysis is required to deduce the coefficients of the polynomial equation.

4.4 Partitioning Algorithm

The partitioning is required when a violation of the constraints is predicted. The OPA performs migrations of tasks from SW to HW in order to decrease the execution time under the time constraint limit. Conversely, when the processing units remain idle before the end of execution of the current period, OPA performs a HW to SW migration in order to free resources from the RCU.

Up-speeding migration. The period (or iteration) time constraint includes the application execution time and the OPA execution time. For a video application, period takes 40ms; this should include the image processing and the OPA execution. If the OPA maximum run time is 1 ms, then the application is allowed to run for 39 ms. So if the application execution is predicted to run for more than 39 ms, then the partitioning must accelerate one or more tasks to reduce that time. In practice we consider a lower limit allowed to the application execution to avoid constraint violation due to errors in estimation and prediction. On line partitioning is performed in OPA by a migration process that changes the allocation of tasks, one task at once, until time constraints are met. Indeed, partitioning is not performed from scratch, it consist of an updating of the current mapping in order to take into account of local variations of execution times. The migration of a task consists of choosing a faster implementation among the set of contexts embedded in the system. The efficiency of the on-line partitioning depends on the right speed-up vs resources implementations selected in an offline profiling analysis and synthesis. For each potential migration the algorithm evaluates the parameter G defined as the product of the time benefit and the number of resources remaining free after migration. To examine the potential migration of the tasks we start by sorting the candidates' implementations by the decreasing order of G (Figure 4). If this up-speeding migration does not satisfy the time constraint then the next candidate migration is performed until the temporal constraint is satisfied or when the reconfigurable is fully exploited. Then we created a version of the application with all the critical tasks moved to the reconfigurable hardware.

Up-freeing migration. This type of migration is necessary to avoid the saturation of the reconfigurable caused by successive SW to HW migrations. The method is analogous to the up-speeding migration. We choose the task which has the minimum benefits of time on the Hardware and that make free the maximum of hardware resources. The partitioner will repeat this process until the total execution time is in between HL (High Level) and the LL (Low Level).

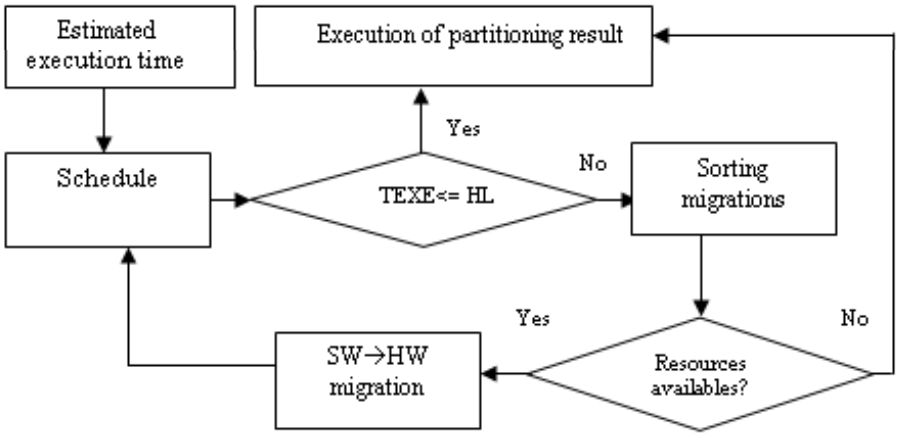

Fig. 4. Up-speeding Migration Approach

4.5 Scheduling Algorithm

To evaluate a partitioning solution, we must calculate the total execution time of the application on that partition. The objective is to find an execution order for the application. Due to the data dependencies in the DFG (described by the edges between tasks) and the sequential nature of the SW processor, the evaluation of the total execution time requires determining a schedule of the DFG. This schedule is based on the estimated execution times of the tasks according to the potential allocation provided after partitioning. Independent tasks allocated to hardware can run in parallel. The migration of task is considered only if there are enough free resources in the hardware. Scheduling decisions are thus related to tasks allocated to the processor. The task can have two states: waiting and scheduled. The waiting state remains until all the task predecessors are scheduled. If a task is to implement on hardware, then it goes in the scheduled state. If it is allocated to software, then the scheduler needs to check if the timing allows the task to be scheduled. There are three possible cases: 1. when the software resource is free and only one software task to be scheduled, 2. when the software resource is free and there is more than one software task to schedule, 3. when the processor is busy. In the first case, the task is scheduled because it is the only one that requires the processor. In the third case, all the tasks wait for the software resource to be free. All this tasks add a new sequencing dependency to their data dependencies. In the second case, the scheduler must choose the software task to be scheduled first. For this a priority table is constructed. The tasks priorities are affected according to the algorithm shown in figure 5. The value of a priority is only based on the successors of the task. The task with most critical successors has the highest priority. A task is critical if it has at least one hardware successor. The highest estimated execution time of all the hardware successors determines the first level of priority. If there is still more than one task having the same urgency then the second level of priority is the task which

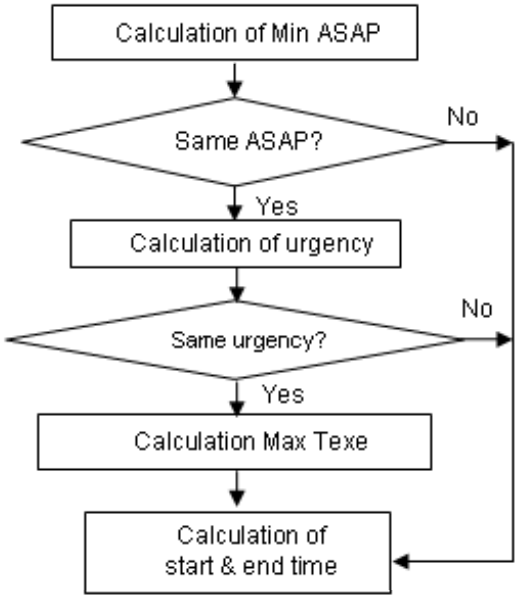

Fig. 5. New HW/SW scheduling approach

has the lowest execution time. In the DFG example shown in figure 6, there are two possible scheduling paths: after scheduling the task A on the processor, we can schedule B or C on the processor. This is because the computation of ASAP (As Soon As Possible) time and the urgency of tasks are as follows: The task B is more urgent than the task C because its successor is the more critical tasks in the DFG. The task D is a hardware task and its execution time is the highest one so its beneficial to schedule it as soon as possible. Given the complexity of the scheduling algorithm, we choose to implement it on hardware. Thus the computation time of the scheduling algorithm is neglected.

The scheduler architecture is fully synthesisable, and it was done with a XILINX Virtex II Pro FPGA tools. The size needed (number of CLB: Configurable Logic Bloc) depends upon the number of tasks, and the complexity of their interconnections. Moreover, the data bus size influences the whole synthesis process as much as the connections routing. To have a clear idea, the synthesis is done with the target core Virtex II Pro vp100: For the Data Bus Size of 10 bits the Whole Scheduler block takes:

- Number of Slices: 1677 out of 44096 3
- Number of Slice Flip Flops: 230 out of 88192 0
- Number of 4 input LUTs: 3088 out of 88192 3
- Minimum period: 11.336ns (Maximum Frequency: 88.211MHz)
- Minimum input arrival time before clock: 5.879ns
- Maximum output required time after clock: 23.875ns
- Maximum combinational path delay: 23.630ns

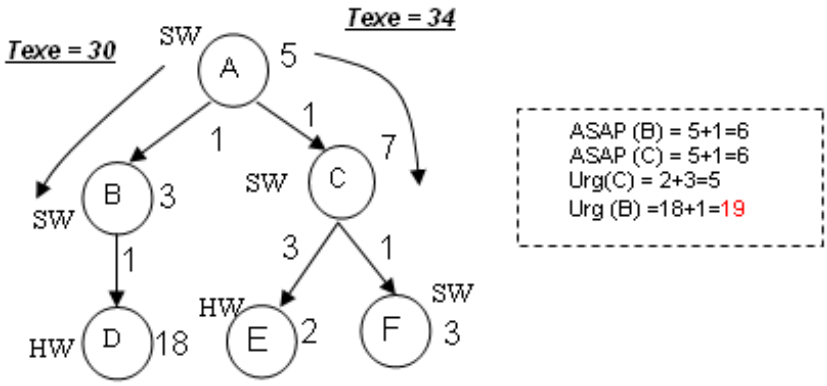

Fig. 6. Scheduling with urgency criteria

The idea of accelerating the scheduler makes the loop Partitioning-Scheduling more feasible in real application. By doing this, the gain of time is great and the cost of hardware is not excessive. All the results are obtained by simulating with Modelsim and Xilinx synthesis tools. The results satisfy the requirements for this OPA, which is minimising the cost of the Estimation-Scheduling-Partitioning time consumption.

5 Experimental Results

We have simulated our partitioning approach with SystemC on an image processing application. The motion object detection on fixed image background requires resources for data processing in order to process images in a real time. ICAM (Intelligent CAMera) is an algorithm developed by the C.E.A, used for embedded camera to detect objects motion (figure 7). Such application can be used for parking supervising, identification, and pieces selection according to the shape The choice of having ICAM as the test application for the OPA can be resumed to its complexity and its variable execution time. The algorithm of detection is based on the difference between the current input and a reference image. If there is an object on the image but not on the reference then ICAM considers it as object in motion. The given ICAM application is sequential on execution, thus we have modified its DFG. This later has been improved by adding virtual tasks running in parallel with the original ones. Each virtual task would simulate task behaviors with a time and resource consuming. Moreover, the new DFG permits to validate the scheduler algorithm and the behavior of the OPA against a complex system. The virtual tasks could have fixed or varying execution time. ICAM is considering as application with twenty tasks as depicted in the figure 7. Each of them could run on software as well as on hardware. The model in SystemC of this application must have the software and the hardware parts. On the software part, each task is a function which is called by the processor when needed. On the hardware side, each task is an independently hardware block

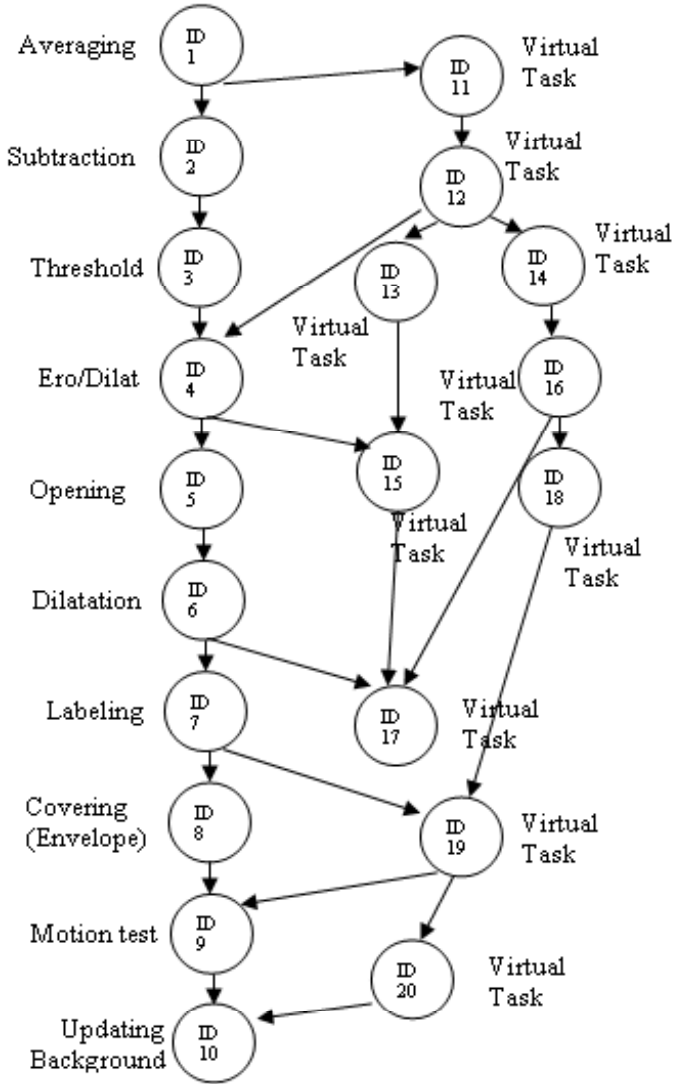

Fig. 7. DFG application

with inputs and outputs for data communications. The application running is controlled by a kernel function, thus the software and hardware parts of the ICAM have a direct communication with the Kernel function. Processed data must circulate from task to task, and this is done by memory communications. It has been considered that the ICAM tasks share the data through a dedicated memory, and then there should have no direct data transmission from software to hardware application tasks.

The communication with memories is only for image data. The communication with the Kernel function is control data: measured execution times and CP values. The execution time is measured with the simulating platform processor clock: the modeling and simulating have been done under a Pentium Centrino 1.5 GHz with MS Windows XP as O.S. The behavior of the system can change with the speed of the processor and the used O.S to run SystemC. Only software execution times are measured. The hardware execution times are deduced from the software by dividing this later over a fixed coefficient. The following table shows the execution time of all OPA functions:

Algorithm	Average execution time (ms)
Partitioning	0.330
Scheduling	0.002
Estimator	0.421
Database updating	0.010
Application	33.565

According to the table above, we notice that the OPA timing cost is neglected as compared with the execution time application. For our experiment the next CP value is computed by adding 5 percent to the precedent one. A comparison between measured and estimated execution time proves the efficiency of the estimator method. Figure 8 shows the execution time variation of the estimator algorithm on various iterations. As mentioned in the table above, this algorithm takes 0.421ms in average.

As depicted in figure 9, the Database management never exceeds the 0.012ms.

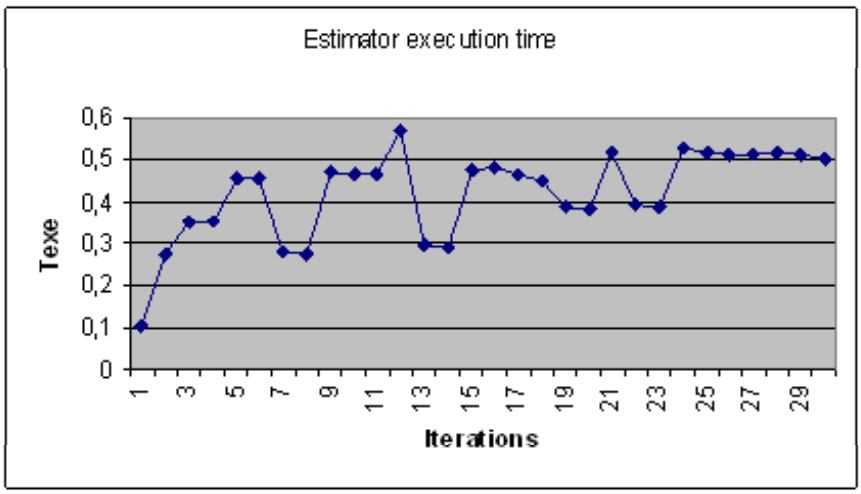

Fig. 8. Estimator computing time per iteration

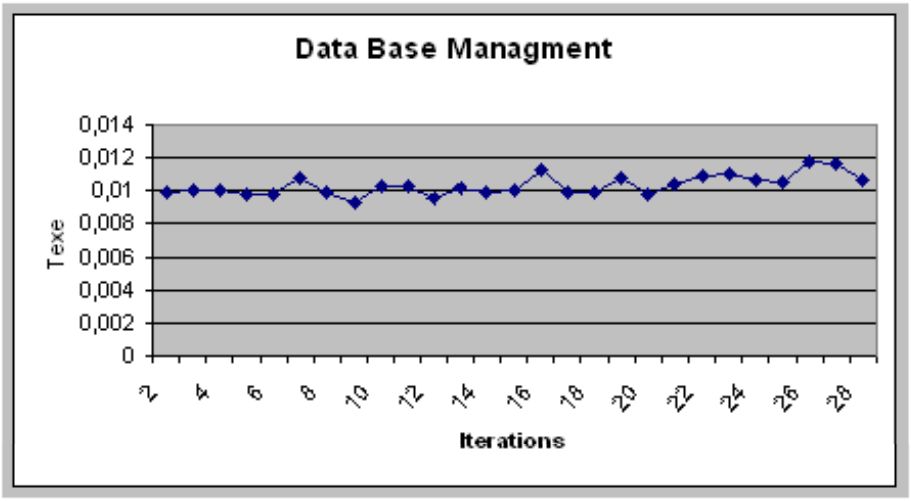

Fig. 9. Time computing of data management function

We remind that the OPA approach has for objective to adapt on-line the partitioning result with the processing requirements. This is shown clearly in the figure 10: the OPA limits always the application execution time between the high level time and the low level time by updating the HW/SW partitioning result.

We compare our partitioning approach with an optimal off-line partitioning method based on ILP (Integer Linear Programming) solver. The ILP based approach can solve the partitioning problem optimally. The partitioning problem is formulated as follows: The objective function is to minimize the number of hardware resources to be used by the application. The main constraint is that the total execution time must respect the real time constraint. The ILP is a static partitioning method based on the Worst Case Execution Times (WCETs) of the application tasks. As it is assumed in OPA, the communication delays between tasks are neglected. The table hereafter shows the differences between the two approaches results:

Approach	Resources used	Nbre Image lost	Off-line work	Resources added	flexible
ILP	852	0	Yes	No	No
OPA	300	0.35 percent	Limited	Yes	Yes

As depicted on the table above, ILP demands more resources for the application than OPA since it tries to reach the optimal solution. The OPA approach tries only to satisfy the constraints.

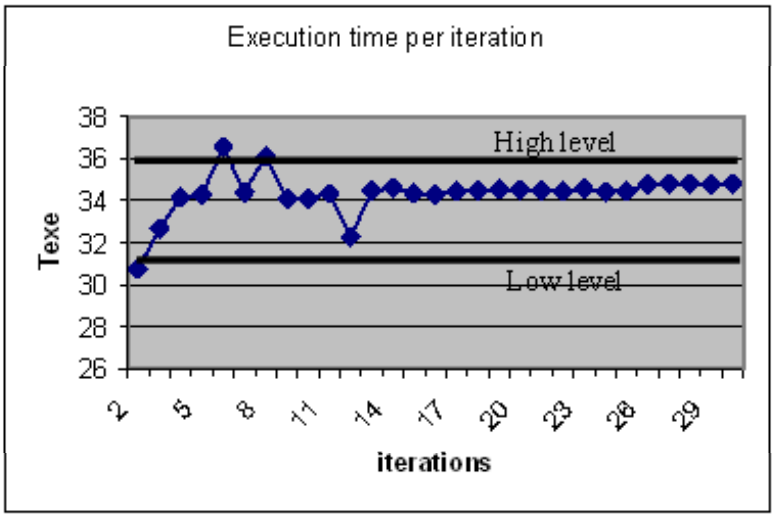

Fig. 10. Application total execution time per iteration

6 Conclusions

Results of our approach show the efficiency of the adaptation of partitioning to needs of treatment. The dynamics reconfiguration of the FPGA allows the architecture to accept several contexts of reconfiguration as results of HW/SW partitioning. A dynamic Hardware /software partitioning approach have many advantages over traditional partitioning approaches. Dynamic partitioning can adapt to an applications constraints dynamically at run time. We presented a HW/SW partitioning approach based on on-line reallocation tasks. We also presented our approach of scheduling based on a criticality parameter chosen on-line. Our future work consists in validating these approaches on real applications and industrial platform.

Acknowledgments. We thank the CEA for their donation of the ICAM application. This work was supported in part by the National Science Research Centre CNRS France.

References

1. Bala, V., Duesterwald, E., Banerjia. Dynamo: a transparent dynamic optimization system. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implemenation, 2000.
2. Balboni, A., W. Fornaciari and D. Sciuto. Partitioning and Exploration in the TOSCA Co-Design Flow. International Workshop on Hardware/Software Codesign, pp. 62-69, 1996.
3. Eles, P., Z. Peng, K. Kuchchinski and A. Doboli. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. Kluwer's Design Automation for Embedded Systems, vol2, no 1, pp. 5-32, Jan 1997.

4. Ernst, R., J. Henkel, T. Benner. Hardware/Software Cosynthesis for Microcontrollers. IEEE Design and Test of Computers, pages 64-75, October/December 1993.
5. Dean, J., Hicks, J., Waldspurger, C.A., Weihl, W.E., Chrysos, G. ProfileMe: Hardware support for instruction level profiling on out-of-order processors, MICRO 1997.
6. Henkel, J., Y. Li. Energy-conscious HW/SW-partitioning of embedded systems: A Case Study on an MPEG-2 Encoder. Intl. Workshop on Hardware/Software Codesign, 1998.
7. Excalibur, Altera Corp., http://www.altera.com.
8. Triscend Corporation, http://www.triscend.com, 2003.
9. Virtex II Pro, Xilinx Corp., http://www.xilinx.com.
10. M. Auguin, K. Ben Chehida, J. P. Diguet, X. Fornari, A. M. Fouilliart, C. Gamrat, G. Gogniat, P. Kajfasz, and Y. Le Moullec, Partitioning and CoDesign tools and methodology for Reconfigurable Computing: the EPICURE philosophy, Proceeding of the third International Workshop on Systems, Architectures, Modeling Simulation (SAMOS03), July 2003, Samos, Greece.
11. Roman Lydecky, Frank Vahid A configurable Logic Architecture for Dynamic Hardware/Software Partitioning. In Proc. of the DATE 2004 Conference. Paris, February 2004.
12. J-Y. Mignolet, V. Nollet, P.Coene, D.Verkest, S.Vernalde, R. Lauwereins Infrastructure for Design and management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-chip. In Proc. of the DATE 2003 Conference. Messe Munich, Germany March 3-7, 2003.
13. Stitt, G., Lysecky, R., Vahid, F. Dyanmic hardware/software partitioning: a first approach. Proceedings of the 40th ACM/IEEE Conference on Design Automation (DAC), 2003.
14. Stitt, G. and F. Vahid. The Energy Advantages of Microprocessor Platforms with On-Chip Configurable Logic. IEEE Design and Test of Computers, Nov/Dec 2002.
15. Pettis, K., Hansen, R.C. Profile guided code positioning. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), June 1990.
16. Henkel, J. A low power hardware/software partitioning approach for core-based embedded systems. Design Automation Conference (DAC), 1999.
17. Calder, B., Feller, P., Eustace, A. Value profiling. MICRO pp. 259-267, 1997.