HW/SW Partitioning Approach on Reconfigurable Multimedia System on Chip

Kais Loukil kais_loukil@yahoo.fr
CES Laboratory/ENIS National Engineering School
University of Sfax
Sfax, 3062, Tunisia

Nader Ben Amor nader.benamor@enis.rnu.tn
CES Laboratory/ENIS National Engineering School
University of Sfax
Sfax, 3062, Tunisia

Mohamed Abid mohamed.abid@enis.rnu.tn
CES Laboratory/ENIS National Engineering School
University of Sfax
Sfax, 3062, Tunisia

Abstract

Due to the complexity and the high performance requirement of multimedia applications, the design of embedded systems is the subject of different types of design constraints such as execution time, time to market, energy consumption, etc. Some approaches of joint software/hardware design (Co-design) were proposed in order to help the designer to seek an adequacy between applications and architecture that satisfies the different design constraints. This paper presents a new methodology for hardware/software partitioning on reconfigurable multimedia system on chip, based on dynamic and static steps. The first one uses the dynamic profiling and the second one uses the design trotter tools. The validation of our approach is made through 3D image synthesis.

Keywords: Partitioning, Design Trotter, Profiling, Reconfigurable System.

1. INTRODUCTION

Today's demanding consumer multimedia applications has created new challenges for system on chip (SoC) developers. Due to their complexity, they require high performance architectures. Their integration in a portable and battery operated system impose stringent constraints about energy consumption and final cost. This makes that dedicated architectures the best choice a priori for their integration. This limits the flexibility of the final product which is an important feature for multimedia systems. Indeed, media applications are often changing due to the norm evolution itself (from MPEG2 to MPEG4 to H.264). The same application in the same norm can be executed with different modes and parameters values. For example MPEG4 can be executed with simple or main profile. On the other hand, the mobility of the target system imposes flexibility that permits to the system the use of different norms depending of its location (Bluetooth or Wifi for example). The download of new services according to the context variation. As those media systems are rapidly evolving, the time to market became more and more shorter witch impose reuse concept. All those constraints make flexible architectures an ideal architectural choice for multimedia systems provided that will be enhanced with powerful multimedia capabilities. Several industrials propose processors and platform for multimedia applications. Those architectures are optimized for the application that were designed for (typically video treatment) and are not necessarily sufficient powerful for incoming or other applications (such as 3D image synthesis).

Some approaches of joint software/hardware design (Co-design) were proposed in order to help the designer to seek an adequacy between application and architecture that satisfies the many design constraints (cost, performance, surface, consumption, flexibility…). Currently, with the rapid evolution of reconfigurable architectures like FPGA, new co-design techniques must be investigated due to the variability of the mobile system [15]. In this paper, we present a new approach for multimedia system design based on mixed static and dynamic strategy.
This paper is organized as follows: the first part is devoted for the presentation of the state of the art and positioning of our work compared to the community. The second part describes our approach for HW/SW partitioning. The third part presents the validation through the 3D image synthesis. Finally, we conclude about our work and present some perspectives.

2. STATE OF THE ART

The goal of profiling is to analyze the program statically and/or dynamically in order to determine relevant information for design exploration, hardware/software partitioning and parallelization. Multiple profiling techniques are developed to analyze the input application behavior for different criteria such as performance, memory, power consumption etc[AP].
The methods proposed in the literature can be classified in three categories: statics, dynamic and mixed [1].
Static method estimate of performance of a design solution using static analysis of an input specification, (example: analyze ways in a specification). Dynamic method permit the measurements of performance of a solution are the result of a dynamic analysis of a specification (suing simulation for example).
Mixed Dynamic/static method: It is the use of some elements of the two preceding approaches for the analysis of performance of a solution [12]. The static approaches are in general very fast, but they present an average precision considering the distance which separates the implementation and the specification [1]. The dynamic approaches are more precise but slower considering the time taken for obtaining both simulation model and time [13]. These disadvantages limit the use of these two methods. The majority of the existing tools for estimation use mixed methods to profit from the advantages of the two preceding methods.
The method described in [3] is based on static analyzes (calculation of the execution time based on the assembler code) and dynamic (profiling). Whereas in [5] [4], they calculate the metric separate ones for the software, the hardware and the communication. Then, they use these metrics in particular equations. For the software part, they calculate the execution time in the worst case by using techniques of analysis of way. [18] [6] estimates the performance by using the execution time techniques on low level for the software, the hardware and the communication. The authors of [7] tackle the problem from a generic point of view. They analyze, on the level system, the interaction between the various processes by giving best and the worst time for each one of them. Then, on the basis of an acyclic graph representing the dependences of data between the processes, they calculate the execution time, in the worst case, for the whole system. In [2],[9] they describe a method of mixed static/dynamic estimation. It is made in two stages. The first one is Pre-estimate: A profiling of the description of the system is carried out to obtain execution times for various levels (process, basic block, communication).
On line estimation: The results obtained during the phase of pre-estimate are used in complex expressions for the calculation of the total performance of the system.
The majority of these tools offer a good estimate for the software part; but, for the hardware part, we always finds values considered different from the actual values: what influences the estimate of the execution time of the complete system considering the distance which separates the specification from the system of the hardware implementation. In more quoted work are based on estimation techniques by high level abstract analysis (analyzes critical ways).
Make tradeoff between the offered criteria. In our methodology, we combine both static and dynamic approaches for efficient profiling tool in terms of accuracy and speed for hardware/software partitioning. During the profiling phase, relevant execution and data storage information are collected. In addition, preliminary estimations of the hardware and the software costs of the application’s kernels are performed.

3. APPROACH

The main goal of hardware/software co-design is to shorten time to market while reducing the design effort and costs of designed products. Moreover, the increasing heterogeneous nature of embedded system platform and application make necessary a good hardware/software partitioning and optimization. Design Space Exploration of embedded system allows rapid performance evaluation of different design parameters and application hardware/software partitioning.
In our work we us two steps methodology Figure 1: profiling and a static analysis using the Design Trotter environment.

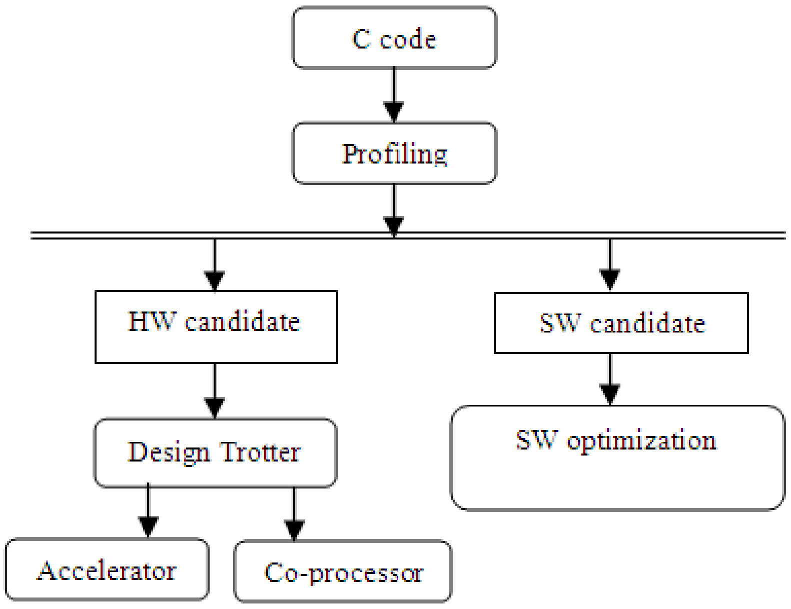

FIGURE 1: Proposed approach

3.1 Profiling

It is obvious that a hardware task is carried out more quickly than software. Thus to accelerate the treatment of a task it is advised to implement it in the form of HW accelerator. In order to determine which part of the application must be optimized of speed of execution, we chose a dynamic analysis of the application software.
Dynamic profiling is a performance analysis technique that measures the behavior of a running program particularly measuring the frequency and duration of function calls. This investigates the program's behavior using information gathered as the program runs [AP]. The output is a stream of recorded events (a trace) or a statistical summary of the events observed (a profile). Dynamic profiling involves obtaining the execution time and percentage call of various functions. This gives us an idea of where the processor is spending most of its time. Dynamic analysis is performed with executing programs. The most called functions are the privileged candidates for hardware implementation. Those HW candidates are them analyzed using the Design Trotter codesign tool.

3.2 Design Trotter tool

The aim of this analysis is the characterization of the first step candidates to detect if they are suitable for HW acceleration. A often called function with many control structure (like while loop) is not suitable for HW acceleration. Functions with intrinsect parallelism are also not suitable for HW accelerations.
Design Trotter (DT) is a set of cooperative tools which aim at guiding embedded system designers early in the design flow by means of design space exploration, as summarized in Figure 2. It operates at a high-level of abstraction (algorithmic-level). Firstly the different functions of the applications are explored separately. For each function, the two first steps (presented in

this paper) include the construction of the graph (HCDFG model) and the characterization by means of metric computations. Then a scheduling step, presented in [10] is performed for all the data-flow graphs and loop nests within the functions. The design space exploration is performed by means of a large set of time constraints (e.g., from the critical path to the sequential execution). Finally, the results are combined to produce trade-off curves (number of resources vs. number of cycles). The scheduling process offers different options including the balance between data-transfers and data-processings and the use of loop unrolling to meet time constraints.[9]

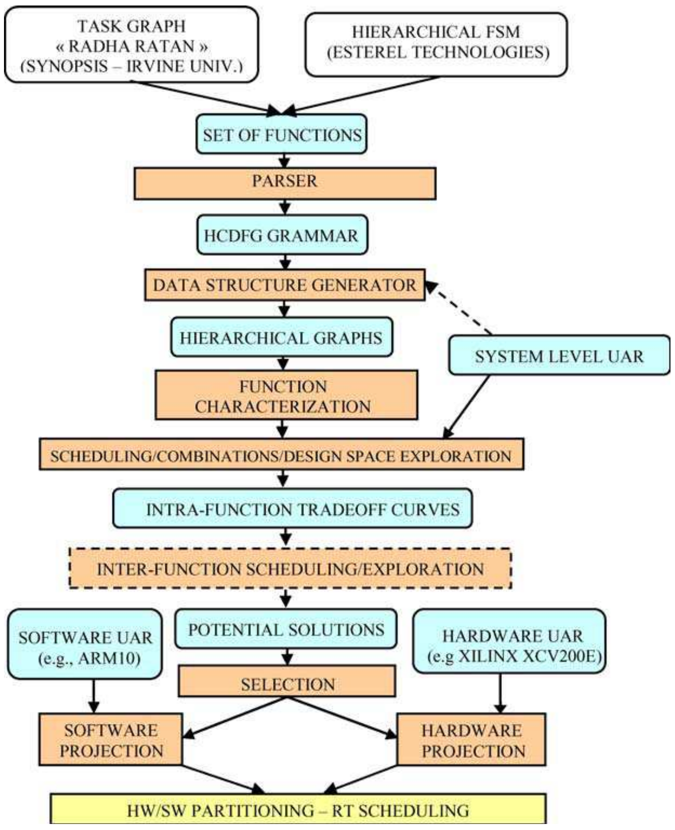

FIGURE 2: Design trotter tools

The entries of the environment Design Trotter are appeared as functions described in C high-level language or using the HCDFG model. (Hierarchical and Control Data Flow Graph)
Each C function of the specification is a node at the top level of the hierarchical control and data flow graph (HCDFG). A function is a HCDFG. A HCDFG is a graph that contains only HCDFGs and CDFGs. A CDFG contains only elementary conditional nodes and DFGs. A DFG contains only elementary memory and processing nodes. Namely, it represents a sequence of non-conditional operations. There are three kinds of elementary (i.e. non hierarchical) nodes of which the granularity depends on the architectural model: a processing node represents an arithmetic or logic operation. A memory node represents a data transfer (memory operation). Its parameters are the transfer mode (read/write), the data format and the hierarchy level that can be fixed by the designer. A conditional node represents a test operation (if, case, loops, etc.) [10].
In our work, we use the “function characterization” module of the DT tool. Four metrics are defined for function orientation characterization. : γ (Parallelism Upper Bound Metric), MOM (Memory Orientation Metric) and COM (Control Orientation Metric).

TABLE 1: METRIC DEFINITION

Metric definition

Metric name	Type of graph	Formula	
γ	General definition	$\gamma = \frac{\text{Nb of global memory accesses and processing operations}}{\text{Critical Path}}$	(1)
γ	IF graph	$\gamma = P_{\text{true}} \frac{\text{Nop}_{\text{true}}}{\text{CP}_{\text{true}}} + P_{\text{false}} \frac{\text{Nop}_{\text{false}}}{\text{CP}_{\text{false}}} + \frac{\text{Nop}_c}{\text{CP}_c}$	(2)
γ	Combination of sequential graph	$\gamma_{\text{equivalent}} = \frac{\sum_i^{\text{total subgraphs}} \text{Nop}_i}{\sum_i^{\text{total subgraphs}} \text{CP}_i}$	(3)
γ	Combination of parallel graph	$\gamma_{\text{equivalent}} = \frac{\sum_i^{\text{total subgraphs}} \text{Nop}_i}{\text{Max}_i\{(\text{CP}_i)\}}$	(4)
MOM	General definition	$\text{Mom} = \frac{\text{Nb of global memory accesses}}{\text{Nb of processing operations} + \text{Nb of global memory accesses}}$	(5)
COM	General definition	$\text{Com} = \frac{\text{Nb of test operations}}{\text{Nb of processing operations} + \text{Nb of global memory accesses}}$	(6)

For a DFG graph g is defined by formula (1) in Table 1. The critical path, noted CP, in a DFG graph, is the longest sequential chain of operations (processing, control, memory) expressed in terms of cycle number. CP is computed for each hierarchical level with a data and control dependency analysis. MOM metric is defined by the general formula (5) in Table 1. MOM indicates the frequency of memory accesses in a graph. MOM values are normalized in the interval. The closer to 1 MOM is, the more the function is considered as data-access dominated. Therefore, in the case of hard time constraints, some high performance memories are required (large bandwidth, dual-port memory, etc.) as well as an efficient use of memory hierarchy and data locality [11]. To calculate this metric, test operations, namely the following operators: <,>, !=, must be identified. COM is defined by the general formula (6) in Table 1. It indicates the appearance frequency of control operations
(i.e. tests that can be eliminated at compilation time) in a graph. The designer can refer to this metric to evaluate the need for complex control structures to implement a function. For example functions with high COM values are most likely to better implemented on a GPP processor rather than on a DSP processor since, the latter is not well suited for control-dominated algorithms. It also indicates that implementing such functions in hardware would require rather large state machines.

4. VALIDATION

4.1 3D synthesis overview

The 3D pipeline is a whole of the necessary stages to the creation and the visualization of a 3D object. This chain is broken up into a whole of necessary operations to post a 3D object observed from a position and with a given orientation. The whole stages are presented in figure 3.

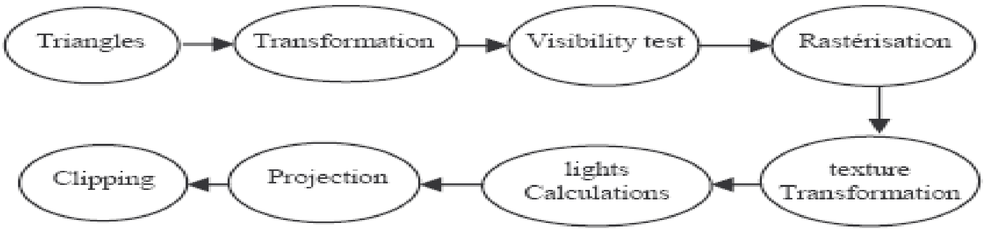

FIGURE 3: 3D graphic pipeline

Transformations: The different transformations consist on translations, rotations and homotheties of the triangles to convert them from their local reference marks to the total scene reference mark and then to the camera reference mark.
visibility test: The visibility test of a triangle is made to determine if it is visible or not.
lights calculations: The models of illumination, taking into account only the interactions between a surface and the source of light, determines the intensity of the color in a point.
textures transformations: This step makes it possible to transform textures before being applied to the triangle in the Rasterisation step.
Clipping: In this step, we eliminate the triangles which do not belong to the volume of sight and we cut out those which have visible parts according to their intersections with the sight volume.
Projection: The projection is the transformation which makes it possible to give the position of the point image on the plan starting from a point in the space.
Rasterisation: The rasterisation is the step responsible for transforming the 3D geometrical forms into pixels on the screen while giving a real aspect to the 3D object in question. In this part, we deal with the shade model.
The figure 4 shows the tasks graph associated with the 3D image synthesizes application. The entry of this graph is a set of local references of the various polygons summits that constitute the 3D object.

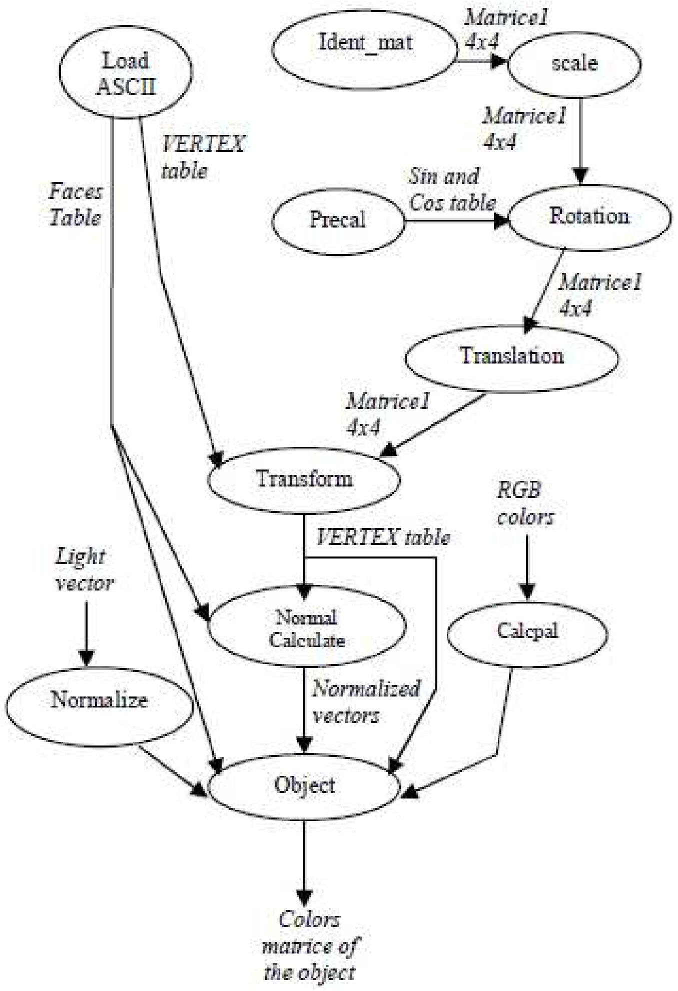

FIGURE 4: Task graph of the synthesis 3D application

4.2 3D Synthesis Application Analysis

To identify the critical (often-called) functions which require a hardware implementation, it is necessary to analyze the 3D synthesis application using profiling technique and high level complexity analysis to each function. As we plan the use of the NIOS embedded processor for the configurations set up, we adopt its associated profiling tool "NIOS IDE". It has a "Performance Counter" that determines the different functions execution time, their percentage as well as the call number of each function. This will enable us to identify the critical functions requiring hardware modules for their execution. Table 2 shows the profiling results of the synthesis 3D application. Using the task Graph in the figure 4, we can identify the most really most called functions. The most critical function is "Object_draw". We analyse them using the "Design Trotter" tool which enables to know more precisely their orientations.

TABLE 2: Profiling result of the 3D synthesis

Section	%	Time(sec)	Occurrences
Transform	6.03	26.875	360
Normal_Calculate	13.5	58.966	360
Translation	0.165	0.726	360
Load_ASCII	0.013	0.0566	1
Object_draw	77.6	340.79	360
Rotation	0.479	2.103	360
normalize	2.23	9.787	12960
scale	0.144	0.62963	360

The hardware candidates tasks are analyzed using a high level co-design tool called "Design Trotter" in order to better analyze their inherent parallelism and orientation (memory, treatment or control). Such analysis is useful to define the most suitable implementation for critical functions so that the application-architecture matching goal can be reached. For example, often-called functions with high inherent parallelism are ideal candidates for hardware accelerators. Critical functions with small local data exchange are ideal candidates for coprocessors. Design Trotter tool analyses of separate functions written in C language and computes three metrics. "Gamma" metric is used for inherent parallelism determination. MOM computes the ratio of data exchanges on the total operations executed by the function. COM metric computes the control operation (do while loops, test operations, switch tests …) on the total operations executed by the function.
Table 3 represents the results of the "Design Trotter" analysis.

TABLE 3: Results OF METRIC BY DESIGN TROTTER

Function	Gamma	MOM	COM
transform	3.8750	0.3939	0.000
Normal_Calculate	3.3244	0.6527	0.0425
Object_draw	1.2429	0.8298	0.0038

According to table 3, we note that the "transformation"," normal_Calculate", functions have a relatively high value of gamma; therefore they have an important average parallelism. "Object_draw" function has a high value of MOM metric and a low value of Gamma metric it is a candidate for implementation in coprocessor.

4.3 Architecture Design

Based system using the NIOS II processor and the AVALON bus figure5.

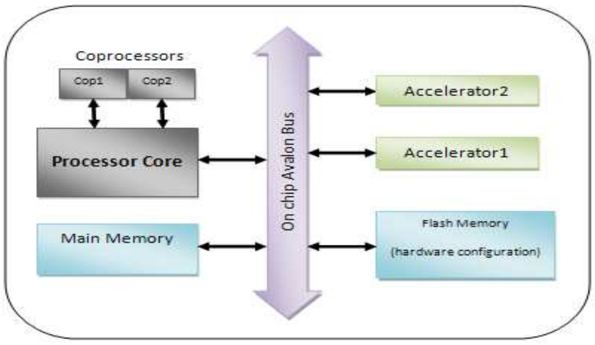

FIGURE 5: Hardware architecture

We realize a system composed of a NIOS II Processor that communicates with HW accelerator through the Avalon bus. The accelerator treats the data while the Avalon Bus supports the control and runs the data tradeoff between HW blocks and memories.
The accelerator communicates with Avalon bus using the “user defined interface”. It is a customizable interface that is automatically generated by the Altera design tool. It supports both slave and master accelerator may.
The slave accelerator (unlike the master one) cannot access directly to the memory to read / write data. It receives data sequentially from the main processor and sends back the result.
The master accelerator can reach the memory via a master port with a DMA mechanism. The HW accelerator receives base address of RAM from the processor via the slave port. It accesses in the external RAM to read data via the master port. Computations are done in combinatorial process. The accelerator accesses again in external RAM via the master port to write the results. This process is repeated until the end of computation. At the end the HW module sends an interruption signal (IRQ) to the processor. Slave accelerators are more ease to design but are slower.
Choosing a method or the other depends on the target application and the desired performance and HW resources occupation. If the processor has to do parallel processing master method is used if else we use the slave method, since it consumes fewer resources on the FPGA.
When the HW accelerator is fully tested, we conduct necessary modifications on the C original code to obtain a new functional code that supports the HW components.

4.4 Accelerator implementation

According to the result of profiling and design trotter tool, we chose to implement (calculates normal, scalar, vector, normalization and transformation) functions as hardware accelerators. For reasons of simplicity the function transformation was implemented using the master method, and other functions are implemented by the slave method.
The figure 6 shows the schematic block of the normal accelerator.

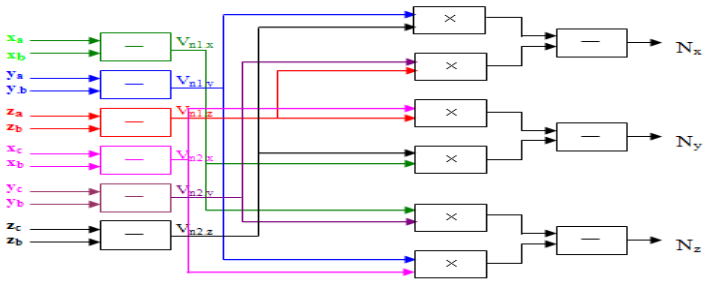

FIGURE 6: schematic block of the normal accelerator

The outputs of the normal calculate circuit can be redirected to the normalization circuit (figure 7) for normalizing the normal vector.

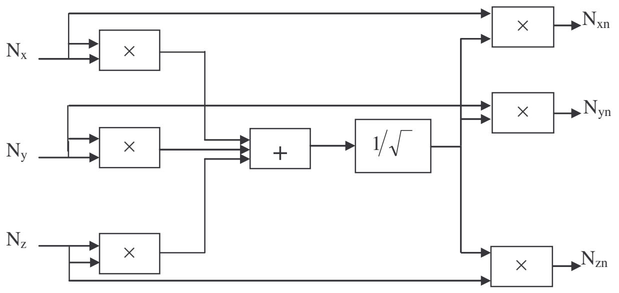

FIGURE 7: schematic block of the normalization accelerator

To ensure the communication between the accelerator and the Nios processor, an interface was developed. Figure 8 describes the overall interface with the signals. This interface consists of different signals I/O accelerators allow to communicate with the rest of the system through the bus Avalon. If we use a slave accelerator, all the master signals can be deleted.

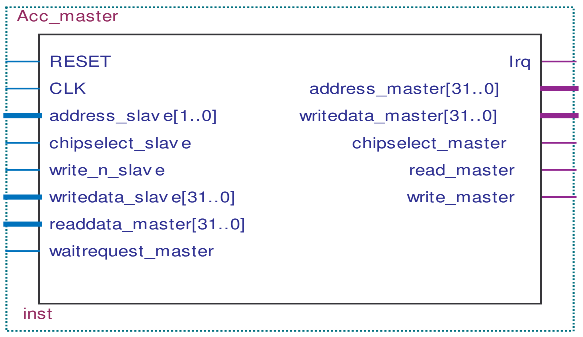

FIGURE 8 : Schematic of the accelerator interface

The following table shows the used FPGA resources for each implemented hardware component. Table 4: ALUT consumed of Hardware component vaut mieux ajouter un pourcentage ou indiquer le total disponible sur le circuit.

Component	**Total ALUT**
CPU+MEM	11075
CPU+MEM+scal	11109
CPU+MEM+vector	11261
CPU+MEM+normalization	15513
CPU+MEM+normal	15793
CPU+MEM+corpocessors	15297
CPU+MEM+coprocessors+normal	19881
CPU+MEM+transformation	14873

4.5 Experimental results

According to the table 3, obtained by the prototyping phase, we notice that the “Object_ Draw” function is the most favored with the hardware implementation. Whereas the results obtained by design trotter, support the functions “Transformation” and “Normal_calculate”. The following table represents the profit of three functions implemented in hardware. We notice that each one of them has an influence over the total execution time of the 3D images synthesis application.

TABLE 5: execution time for 3d synthesis application executed with different architectures

	Execution time	Benefit %
Transformation SW Normal_calculate SW Object_Draw SW	556.032	0
Transformation HW Normal_calculate SW Object_Draw SW	539,742	3%
Transformation SW Normal_calculate HW Object_Draw SW	507,36	9%
Transformation SW Normal_calculate SW Object_Draw HW	400,954	28%
Transformation HW Normal_calculate HW Object_Draw SW	491,07	12%
Transformation HW Normal_calculate SW Object_Draw HW	384,664	37%
Transformation SW Normal_calculate HW Object_Draw HW	352,282	31%
Transformation HW Normal_calculate HW Object_Draw HW	335,992	40%

We note that if we use only the two functions "transformation" and "Normal_Calculate" in the form of hardware accelerator (result of design trotter) we attend 12% of the benefit, while the "Object_draw" function (result of profiling) offers a profit of 28%. In other hand, combining the two solutions offered by the two methods gives a profit of 40%.

5. CONCLUSION

This article deals with the problem related to the hardware/software partitioning of reconfigurable multimedia system on chip. We proposed a new method of HW/SW partitioning. Indeed, our method is based on a static/dynamic mixed approach. For the static part, we used the design trotter tool and for the dynamic part we used the dynamic profiling of the application.
We presented all the steps used for the validation of the method by using the environment of Altera and the 3D images synthesis application.
This method presents interesting results. However, it can be refined by taking account of the data exchanged between the tasks and of the parallelism with the level of the accelerator.

6. REFERENCES

1. A .Baghdadi, exploration et conception systématique d'architectures multiprocesseur monopuces dédiées à des applications spécifiques, thèse PhD, Mai 2002, TIMA France.

2. D.Gajski, F.Vahid, S.Narayan and J.Gong System-level Exploration with SpecSyn. Design Automation Conference, Juin 1998.

3. H.J.Eikerling, W.HARDT, J.Gerlack, W.Rosenstiel: A Methodology for Rapid Analysis and optimization of Embedded Systems. International IEEE Symposium and workshop on ECBS, D-friedrichshafen, Mars 1996

4. J.Grode and J.Madsen Performance Estimation for Hardware/Software Codesign using Hierarchical Colored Petri Nets. Proceedings of Jigh Performance Computing'98, in Special Session on Petri Net Applications and HPC, Boston, Avril 1998.

5. J.henkel and R. Emst, High-level Estimation Techniques for usage in hardware/software codesign. Asia and south Pasific Automation Conference Yokohama, Japan, Fevrier 1998

6. J.Liu, M.Lajolo and A.Sangiovanni-Vincentelli, Software Timing Analysis Using HW/SW Cosimulation and Instruction Set Simulator. International Workshop on Hardware-Software Codesign, Mars 1998.

7. T-Y.Yen and W.Wolf, Communication Synthesis for Distributed Embedded Systems. International Conference on Computer-Aided Design,1995.

8. S. Rouxel Modélisation et Caractérisation de Plates-Formes SoC Hétérogènes : Application

9. Y.le moullec, J.P.Diguet, N. Ben amor, T.gourdeaux, and J.L.Philippe: Algorithmic-level Specification and Characterization of Embedded Multimedia Applications with Design Trotter. Journal of VLSI Signal Processing 42, 185–208, 2006

10. N. Ben Amor, Y. Le Moullecc, J. P. Diguet, J.L. Philippe, M.Abid: Design of a multimedia processor based on metrics computation. Advances in Engineering Software 36 (2005) 448–458

11. S. Wuytack, J.P. Diguet, F. Catthoor, H. De Man. Formalized methodology for data reuse exploration for low-power hierarchical memory mappings. IEEE Trans VLSI Syst 1998;6(4):529–37.

12. K. Bertels, G. Kuzmanov, E. M. Panainte, G. N. Gaydadjiev, Y. D. Yankova, V. Sima, K. Sigdel, R. J. Meeuws, and S. Vassiliadis, "Profiling, compilation, and hdl generation within the hartes project," in FPGAs and Reconfigurable Systems: Adaptive Heterogeneous Systems-on-Chip and European Dimensions (DATE 07 Workshop), April 2007, pp. 53–62.

13. A. Srivastava and A. Eustace, "Atom: a system for building customized program analysis tools," SIGPLAN Not., vol. 39, no. 4, pp. 528–539, 2004.

14. R. J. Meeuws, Y. D. Yankova, K. Bertels, G. N. Gaydadjiev, and S. Vassiliadis, "A quantitative prediction model for hardware/software partitioning," in Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL07), August 2007, p. 5.

15. K. Loukil, N. Ben Amor, M. Ben Saïd, M. Abid, "OS service update for an online adaptive embedded multimedia system," IEEE Symposium on Computers and Communications (ISCC'09), Sousse, Tunisia, Jul. 5-8, 2009