Hardware Parallel Architecture of a 3D Surface Reconstruction: Marching Cubes Algorithm

MILI Manel, MAHMOUD Bouraoui, AKIL Mohamed, BEDOUI Med Hédi

***Abstract*—** In this paper we present a study of the algorithmic and architectural exploration methodology for a parallelism of the 3D reconstructing algorithm (Marching Cubes) and its optimized implementation on FPGA.
We aim at defining a parallel multiprocessor architecture implementing this algorithm in an optimal way and Elementary Processor (EP) architecture dedicated to this algorithm.
We use the SynDEx tool which adapts the AAA (Algorithm Architecture Adequacy) methodology, to find a good compromise between the computing power, the functionality of each PE, the optimization constraint (time, area), and the parallelization efficiency. Then, we describe a first implementation of PE on FPGA

***Keywords*—**3D reconstruction, Marching Cubes, Optimization AAA, FPGA, SynDEx.

I. Introduction

The 3D reconstruction consists in providing a volumetric representation of an object from a set of information that ensures a description of the 3D volume. The involvement of these algorithms is important in various fields, especially in medicine and biology.

The *Marching Cubes* algorithm is the most used for the isosurface reconstruction [1]. This algorithm has been designed by William E. Lorensen and Harvey E. Cline in 1987 [5] to generate a 3D model for an interesting anatomical structure. For that, it uses a threshold (characteristic value of anatomical structure for studying a human organ, which will be defined by the medical expert; each organ has its proper threshold) and a set of slices previously segmented [2] [3].

We have known that the application implementation in processing and 3D reconstruction images must often respect real-time execution, while minimizing resource consumption when targeting systems with low cost and which are able to integrate the maximum processing algorithms.

Our goal is to make the hardware implementation of a fast and robust 3D reconstruction by defining a parallel architecture for the *Marching Cubes* algorithm. This requires the adoption of an algorithmic and architectural optimization methodology as the *AAA* (Algorithm Architecture Adequacy) for rapid prototyping associated with the *SynDEx* tool (Synchronous Distributed Executive) [4]. This tool gives us a well algorithmic and architectural exploration in function of the optimization constraints (time and area), the processing element number and the multiprocessor architecture topology.
In this paper, our work is based on the tool and methodology for studying and exploring the different parallelism types, to define the architecture topology and to specify their elementary processors dedicated to the 3D reconstruction (*Marching Cubes*) algorithm.

This paper is organized as follows: Section 2 describes the *Marching Cubes* algorithm. In Section 3 we describe the *AAA* methodology and the *SynDEx* tool applied for the exploration and implementation of this algorithm, as well as the results of this architectural exploration of parallelism and the corresponding topology. We propose in Section 4 the Elementary Processor architecture and we present its implementation on two families of *FPGA*.

Manuscript received June 6,2012: Revised version received .., 2012. This work was supported in Research Team of Medical and Imaging and Technologie at the Faculty of Medecine of Monastir, Monastir University, Tunisia.

MILI Manel is with Faculty of Medicine/Laboratory of Biophysique/Team TIM, University of Monastir, 5019, Tunisia (e-mail: mili_manel@yahoo.fr).
MAHMOUD Bouraoui is with Faculty of Medicine/Laboratory of Biophysique/Team TIM, University of Monastir, 5019, Tunisia (e-mail: Bouraoui.mahmoud@fsm.rnu.tn).
AKIL Mohamed is with ESIEE Paris/Laboratoire d'informatique Gaspard-Monge/équipe A3SI, Université Paris EST Bld Blaise Pascal, BP 99, Noisy-Le-Grand, 93162, France (e-mail: Akilm@esiee.fr).
BEDOUI Med Hédi is with Faculty of Medicine/Laboratory of Biophysique/Team TIM, University of Monastir, 5019, Tunisia (e-mail: Medhedi.bedoui@fmm.rnu.tn).

II. Marching Cubes (MC)

The principle basic of the "*Marching Cubes*" algorithm is to subdivide the space into elementary volumes [10]. The basic element is a cube called voxel and formed by 8 vertices and 12 edges. Each vertex can get two states: it can be inside or outside the interested surface (surface of anatomical structure). So, there are 256 (2^8) possible topologies inside a voxel. Due to the rotation symmetry and inversion of inner and outer points, these 256 initial configurations can be reduced to 15 basic configurations [5] [10].

This algorithm is functioning is composed by 4 stages: The first step is to define a cube and number its vertices according to the Paul Bourke convention [5]. The second consists in determining the index (Fig.1). In fact, the interested surface intersects the voxel edge when the two vertices forming this edge are two opposite signs. In the third step, this index is used as a pointer in the "*Tri-Table*" (Fig.1) which defines the set of intersections of the interested surface by the cube edges. The last step allows the intersection points calculation on the cube's edges, by a linear interpolation [10] (Fig.1).

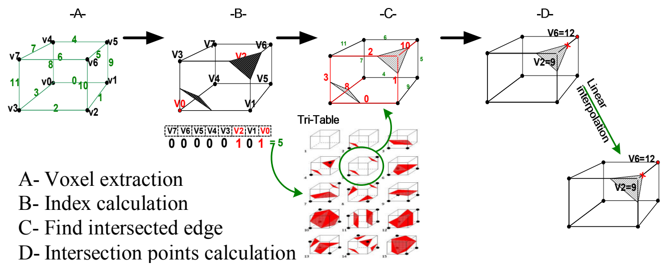


Fig. 1 Steps of the *MC* algorithm's functioning

Algorithm 1 describes the iterative form of the Marching Cubes algorithm, and it takes as data: **p** as the number of 2D images, **N** as the resolution of 2D images (image NxN), and the **threshold** (chosen by the user to be associated to the interest anatomical structure).

The core processing algorithm contains three loops: `(for(s=0; s<7; s++))` and the two nested loops `(for(a=1; a<15; a++))` and `(for c=x;y;z)`.

This algorithm gives, on each voxel processed, the coordinates of the intersection point between the edge and the interested surface (**P_x,P_y,P_z**).

Algorithm 1 :

```
{Data= p slices(N x N), threshold}
for(z=1; z<p; z++)// read of two adjacent slices//
  for(y = 1 ; y<N; y++)
    for(x = 1; x<N ; x++)//Voxel Extraction//
      for(s = 0; s<7; s++)// Index calculation
        If ($V_{xyz}$(s) < threshold) then
        index = index +$2^s$
      for (a = 1; a<15; a++)
        If Tri-Table[Index](a)≤11 then
        //the edge a is intersected, coordinates
        and intensities extraction of pixels
        associated in every edge:
        $V1_{xyz}$,$V2_{xyz}$,($P1_x$,$P1_y$,$P1_z$)et($P2_x$,$P2_y$,$P2_z$)//
        for c = x;y; z
          If(threshold-$V1_{xyz}$)>0 and ($V2_{xyz}$-$V1_{xyz}$)>0
          Then //interpolation calculation//
          else  Pc =P1c
      {Results=$P_x$,$P_y$,$P_z$} //the intersection points//
```

III. Parallelism Exploration Methodology

A. AAA/SynDEx tool

The *SynDEx* (Synchronous Distributed Executive) [6] [7] is a graphic software used to provide rapid prototyping and optimizing the implementation of real-time embedded applications on multiprocessor architectures.

This tool is based on the *AAA* methodology [8]. It takes as an input the algorithm specification in the form of a data graph and the target architecture description in the form of an operator graph (Fig.2).

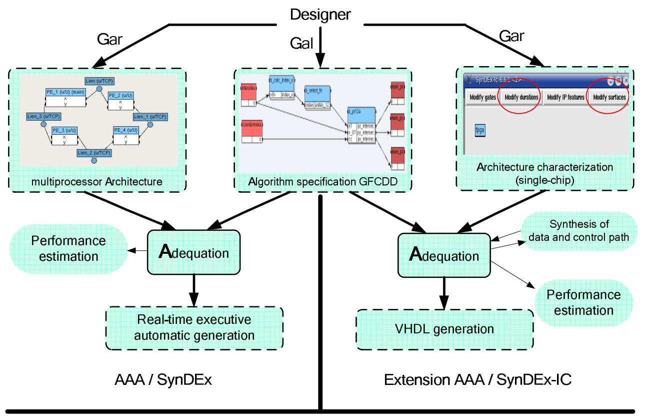

Designer
Gar
Gal
Gar
multiprocessor Architecture
Algorithm specification GFCDD
Architecture characterization (single-chip)
Performance estimation
Adequation
Adequation
Synthesis of data and control path
Performance estimation
Real-time executive automatic generation
VHDL generation
AAA / SynDEx
Extension AAA / SynDEx-IC

Fig. 2 the extension of the AAA / SynDEx

The *SynDEx* tries to make an algorithm implementation which respect a given constraint (latency and surface) while being implemented on the target circuit. For this, it works on factorization; i.e, the more the factorized loop is, the more parallelism is [11].

The factorization process shows four specific types of nodes present in Fig.3 (factorization frontiers nodes): **F** (Fork node), **J** (Join node), **D** (Diffuse node) and **I** (Iterate node).

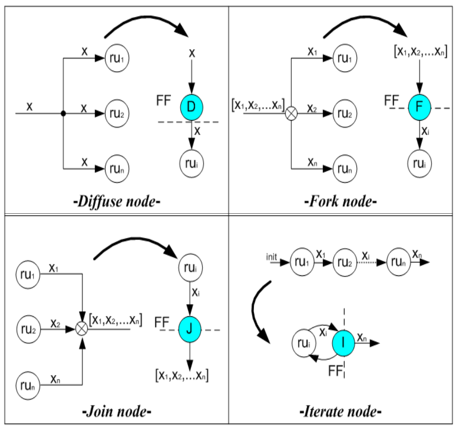


Fig. 3 Factorization frontier nodes

Because it is impossible to explore all possible defactorizations in reasonable times, *SynDEx* use approximate methods based on heuristics.

The *SynDEx* executes the optimization heuristic, adequating between the algorithm and the architecture. This ensures the automatic generation of a real-time distributed executive.

The optimized implementation called adequacy, forms a graph obtained by transforming the two input graphs (algorithm graph, architectural graph).

Among all possible transformations, the heuristic optimization, based on performance prediction, keeps the one that minimizes the execution time (latency) of the algorithm. The result is shown through the temporal graph provided by the *SynDEx*: the "*Schedule Time*" (Fig. 4). This prediction is used for viewing the parallelism obtained as well as optimizing the implementation. For this, the user may interact with heuristics to help find better results by adjusting the grain size of the algorithm and by modifying the resources of the architecture.

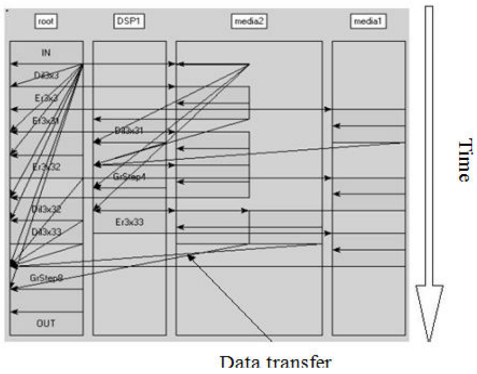


Fig. 4 *Schedule Time*

Following the construction of the implementation graph, the *SynDEx* generates a macro code which does not depend on the language used by processors. The process leading to the final goal of the *AAA* methodology (the algorithm execution on real architecture) includes, at first, the transformation of each macro code constituting the program for each processor in a compilable executive. That is why, it is necessary to have translation libraries given with the *SynDEx* [4].

The *SynDEx-IC* is also a tool like the *SynDEx*; it looks for an algorithm implementation which respects a given constraint (latency, surface) for optimization. However this tool (*SynDEx-IC*) aims for an implementation on a single *FPGA* architecture. Thus, it allows the generation of the synthesizable and optimized *VHDL* code while respecting the constraints.

B. *Modeling of the Marching Cubes*

In this section, we show how we have modeled the *Marching Cubes* algorithm to exploit the different types of parallelism.

1) *Data Parallelism*

In order to optimize memory used on embedded processors, we have analyzed the data dependencies presented in the Marching Cubes algorithm.

As we have described in algorithm 1 in section 2, the 3D reconstruction is done by sweeping each two successive slices to extract a finite number of cubes. Thus, the treatment is done cube by cube. This sweep is repeated for the other slices until the volume formation whose parameters are the resolution of 2D slice (N x N pixels) and the number of slices Z.

In fact, the cube processing presents the working core of this 3D reconstruction algorithm. This treatment is independent from others, not only within the same 2D slice but also by moving from a 2D slice to another, so we can treat many cubes in parallel as a cube block.

As we have outlined below, the processing volume depends on two factors; the image resolution (N) and the slice number (Z), therefore the number of processing blocks can be presented in two ways:

- Version 1: The processing block depends on the 2D image resolution (N): The blocks number $n_b = k^2 \times (Z-1)$ where k presents the parallelism factor. Each block contains $\left(\frac{N-1}{k}\right)^2$ cubes where k= 2^i and $i \in \mathbb{N}^*$.

In Fig.5 we show this approach of two 2D slices. The maximum data parallelism is attained when all cubes extracted from the 2D slices are processed in parallel. In this case, each cube is a processing block.

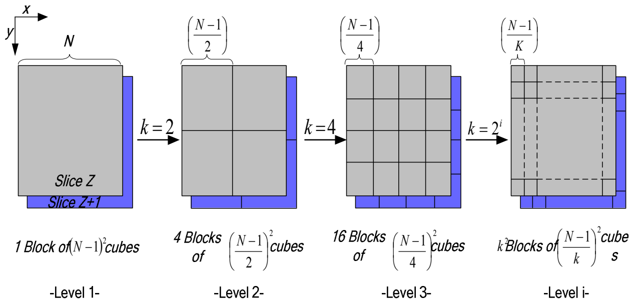

x
y
N
$\left(\frac{N-1}{2}\right)$
$\left(\frac{N-1}{4}\right)$
$\left(\frac{N-1}{K}\right)$
Slice Z
Slice Z+1
$k=2$
$k=4$
$k=2^i$
1 Block of $(N-1)^2$ cubes
4 Blocks of $\left(\frac{N-1}{2}\right)^2$ cubes
16 Blocks of $\left(\frac{N-1}{4}\right)^2$ cubes
k^2 Blocks of $\left(\frac{N-1}{k}\right)^2$ cubes
-Level 1-
-Level 2-
-Level 3-
-Level i-

Fig.5 Data flow model for the calculation of blocks configured by N

- Version 2: The processing block depends on the slice number to process (Z): In this case, the blocks number $n_b = k$ where $1 \le k \le (z-1)$. Each block is composed by $\left[\left(\frac{Z-1}{k}\right)+1\right]$ slices and contains $\left(\frac{Z-1}{k}\right) \times (N-1)^2$ cubes.

In Fig.6 we show this approach.

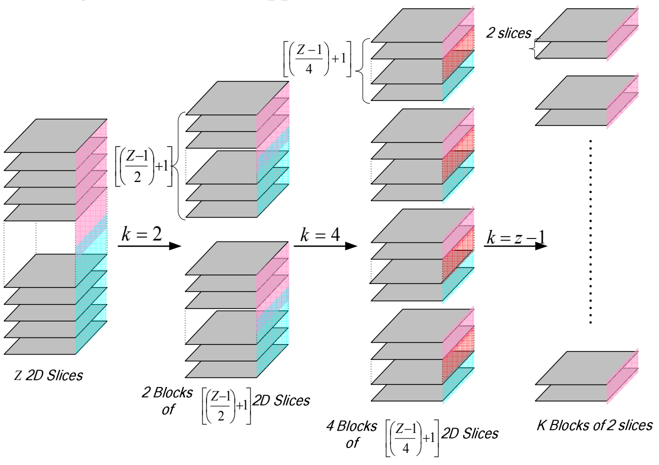

$\left[\left(\frac{Z-1}{4}\right)+1\right]$
2 slices
$\left[\left(\frac{Z-1}{2}\right)+1\right]$
$k = 2$
$k = 4$
$k = z-1$
Z 2D Slices
2 Blocks of $\left[\left(\frac{Z-1}{2}\right)+1\right]$ 2D Slices
4 Blocks of $\left[\left(\frac{Z-1}{4}\right)+1\right]$ 2D Slices
K Blocks of 2 slices

Fig.6 Data flow model for the calculation of blocks configured by Z

The maximum data parallelism is attained when all 2D slices are processed in parallel. In this case, each two successive 2D slices are a processing block.

2) *Complexity analysis*

In the following, we focus on exploring and developing the data flow model configured by N, which we have outlined in the previous section.

According to this model, the complexity calculation is based on the parallelism factor (K).

In the table below we present the complexity of the *Marching Cubes* algorithm which processes 24 2D images (64x64). This complexity depend, on three parameters: the cubes number to be processed per block, the operations number and the memory space occupied.

Slices number (Z)	Resolution (N)	k	Voxel number per block	Operations number per block	Memory space (Koctet)
24	64	2	45643	17892252	89,15
		4	22821	8946126	
		8	11410	4473063	

Table I. Complexity of the Marching Cubes algorithm

The complexity analysis presented in table.I confirms that the voxels number to be processed per block as well as the operations number is depends on the parallelism factor k.

3) Task Parallelism

The *Marching Cubes* algorithm consists in drawing a polygon which presents the intersection of a cube and the plane that models the interest surface, so the core of this processing focuses on a voxel. In section 2 we describe how this algorithm is functioning. We are interested, in the next part (section III.C), in the last stage and we try to put in parallel the calculation of intersection points on the 12 edges of the voxel.

C. Architectural exploration result of the MC parallelism

1) Algorithmic architectural specification

Using the *SynDEx-IC* tool, the algorithm specification is in the form of a data dependency graph including regular (repeating periodic units) and no-regular parts. This specification should be independent of any constraints linked to the hardware implementation and requires the implementation of the algorithm's decomposition process in hardware operations (addition, subtraction and multiplication) [9].

In the example of the second step operation of the *Marching Cubes* algorithm, which consists in determining the index, the value is presented in 8 bits. So it is between 0 and 255. The index value provides information on the positions of the inside and outside for the interested surface. Each bit of the index is associated with one vertex; the bit is "1" if the point is internal, and "0" if is not. The surface then cuts the edges of the cube when the two vertices forming the edge are opposite signs (one is 0 the other is 1).

Thus, by factoring the index calculation block (Fig.7), we have a factorization frontier **FF2** which corresponds to the repeating units factorization of the index calculations (comparator, multiplier and adder) contained in the loop (For (s = 0;s<7 ;s++)).

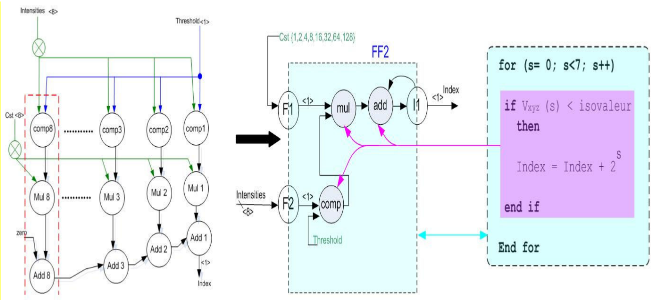
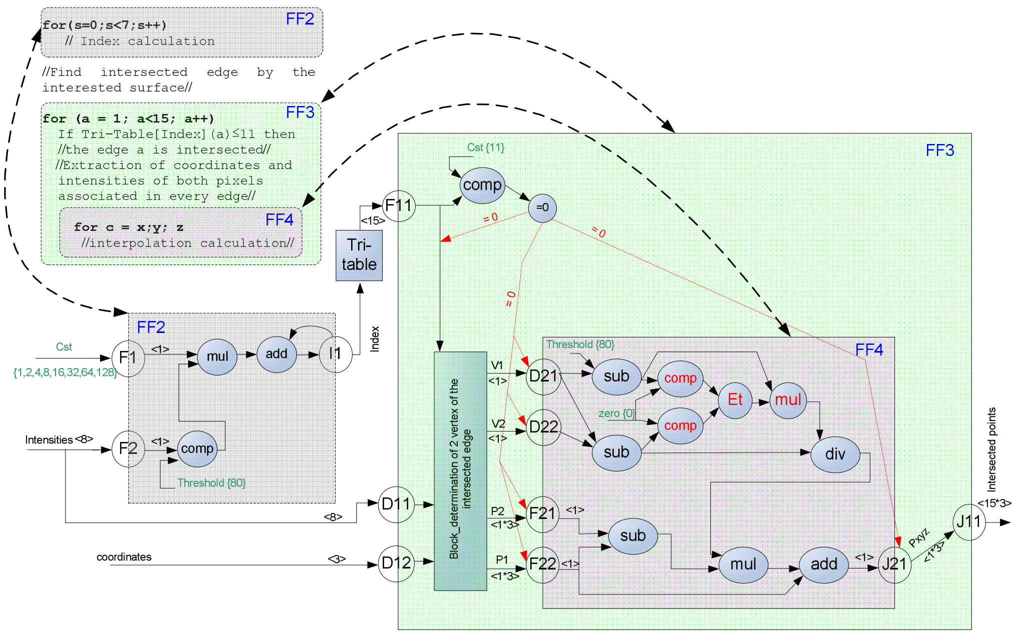


Fig. 7 Decomposition and Factorization of an index calculation block

This **FF2** frontier is applied 8 times and is delimited by the factorization node (**F1, F2, I1**). The two nodes **F1** and **F2** separate their input array into an element of 8 intensities for **F2** and 8 constants for **F1**, while the node (**I1**) makes the factorization of inter-pattern data (Fig.8).

In Fig.7, the *GFCDD* of the *Marching Cubes* algorithm, added to the **FF2**, contains two nested frontiers, **FF3** and **FF4**, whose factorization factors are respectively 15 and 3.

The first is the factorization of unit calculations contained in the loop "((`for (a=1;a<15;a++)`)"; and the second is the factorization contained in the loop "(`for c=x;y;z`)" (Fig.8).



```
for(s=0;s<7;s++)
   // Index calculation
```

```
//Find intersected edge by the
interested surface//
```

```
for (a = 1; a<15; a++)
  If Tri-Table[Index](a)≤11 then
  //the edge a is intersected//
  //Extraction of coordinates and
  intensities of both pixels
  associated in every edge//
    for c = x;y; z
     //interpolation calculation//
```

Fig. 8 *GFCDD* of *Marching Cubes* algorithm for a voxel processing

2) *Setting a graph within SynDEx-IC*

We have modeled the *GFCDD* of the *MC* algorithm in *SynDEx-IC*. The solutions obtained by the heuristic optimization under different constraints (expressed in micro seconds) are given in Fig.9. For each case, we give the estimated latency and area (number of **CLBs**) and the defactorization degree (**DF**) of the two frontiers **FF3** and **FF4** (**TF** indicates that it is totally factorized).

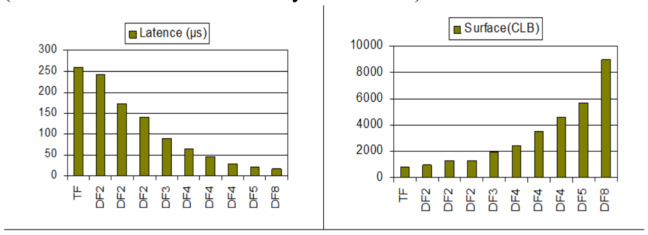


Fig. 9 Results of the heuristic according to constraint variation

The parallelism depends on the degree of each defactorization frontier following the constraint. For a constant surface, the variation of latency depending on the defactorization degree is due to the sequentially or partially parallel execution of the frontier.

According to the results presented in Fig.9, the heuristic has opted for implantation when the frontiers (**FF3** and **FF4**) are defactored by 4; this is confirmed by the neighborhood graph generated automatically by the *SynDEx-IC* (Fig.10).

In this graph, each node represents a factorization frontier and each hyperarc represents the data dependencies between the different frontiers.

The first neighborhood graph (Fig.10a) corresponds to a sequential implementation of the Marching Cubes algorithm. Whereas, the second (Fig.10b) corresponds to its optimized implementation to defactorize the different frontiers by 4 in order to minimize its critical path.

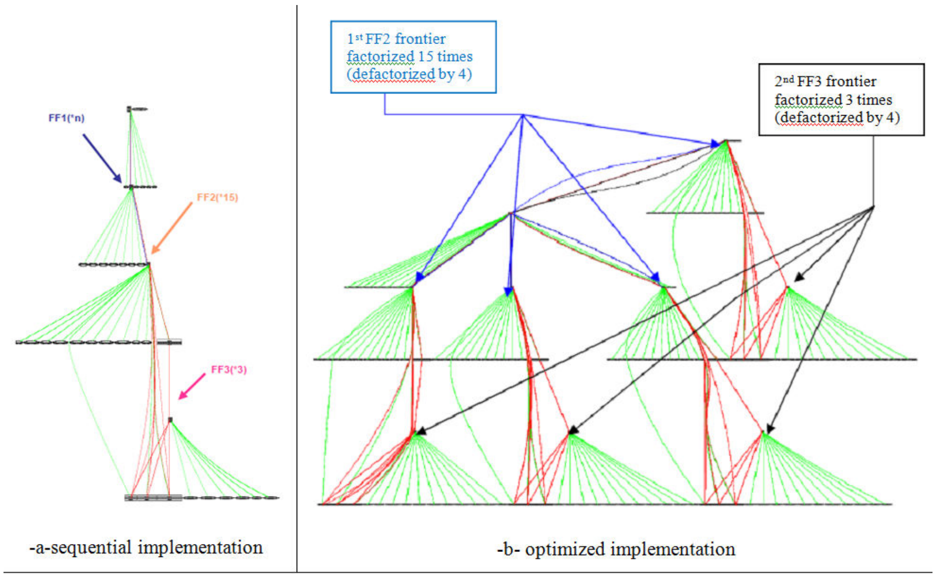

FF1(*n)
FF2(*15)
FF3(*3)
1st FF2 frontier factorized 15 times (defactorized by 4)
2nd FF3 frontier factorized 3 times (defactorized by 4)
-a-sequential implementation
-b- optimized implementation

Fig. 10 The *Marching Cubes* neighborhood

These new frontiers can be executed in parallel (that means to perform processing for each three edges together); this result can provide the reduction of the computation time but increases of the resource consumption.

D. Exploration result of the multiprocessor topology:

Then, we use the *SynDEx* tool to model the data parallelism (version1) in various models of different topology architectures.

We focus in this article on a ring architecture which has led to better results by varying the number of **PEs** (**N_{PE}**). In this architecture, each **PE** is connected to its two neighbors by a direct link (Fig.11).

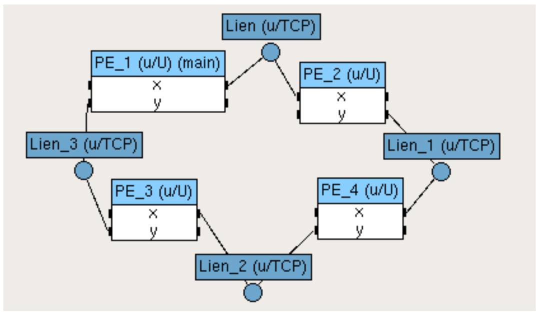


Fig. 11 Example of ring architecture

We have already seen in section III.B.1 that the processing block depends on the two parameters N and Z. According to version1, we consider that the blocks number $n_b = k^2 \times (Z-1)$ and each one contains $\left(\frac{N-1}{k}\right)^2$ voxels where k=2^i, $i \in \mathrm{N}^*$

The exploration has been carried on 24 images of 64x64 as data, by varying two parameters:

- The number of processing blocks (by varying k)
- The number of elementary processors (**PEs**) 1 <**N_{PE}** <64

In fact, the acceleration (Fig.11) is described by S_p (S_p=T_{seq}/T_p) where T_{seq} and T_p are respectively the computation time of 1 and **N_{PE} PEs**.

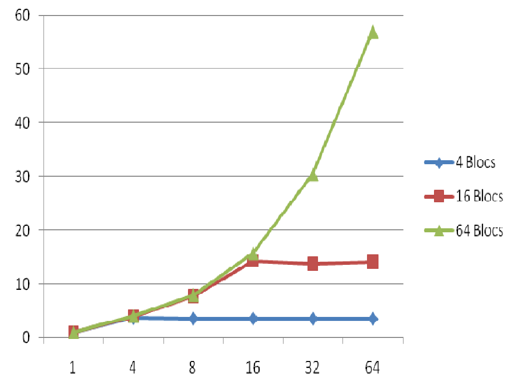


Fig. 12 The acceleration as a function of N_{PE}

The result shown in Fig.12 confirms that the speed of the algorithm continues to increase proportionally with the number of **PEs** processors present in architecture until a defined number, and then it becomes stable.

Indeed, the acceleration reaches its maximum for a multiprocessor architecture whose number of **PEs** is similar to the number of processing blocks. For the case of 16 input blocks, the best compromise between the acceleration and the **N_{PE}** (surface) is attained for 16 **PEs** (k = 4).

According to the data parallelism exploitation for this algorithm, we can conclude that an efficient architecture is the one that forms a ring of **PEs**, where each one processes a single block (each voxel presents a block).

Fig.13 represents the parallelization efficiency (**E_{ff}=S_p/N_{PE}**) of the adequacy obtained by *SynDEx*, depending on the **N_{PE}**.

This efficiency decreases by increasing the number of **PEs** forming the architecture; this may be explained by the fact that this parallelization requires many data transfers between processors.

We can observe in the previous figure (Fig.13) that the decreased efficiency is increasingly important in the case of the architecture that includes a significant number of **PEs** and that processes a few input blocks.

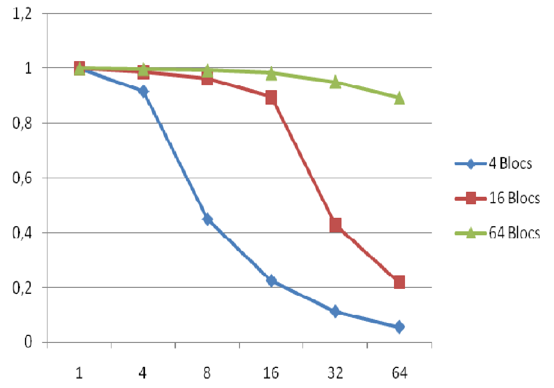


Fig. 13 The parallelization efficiency as a function of N_{PE}

This observation confirms that the addition of **PEs** in an efficient architecture can quickly reduce the efficiency of parallelization, since the data transfer time between processors becomes important compared to the process time.

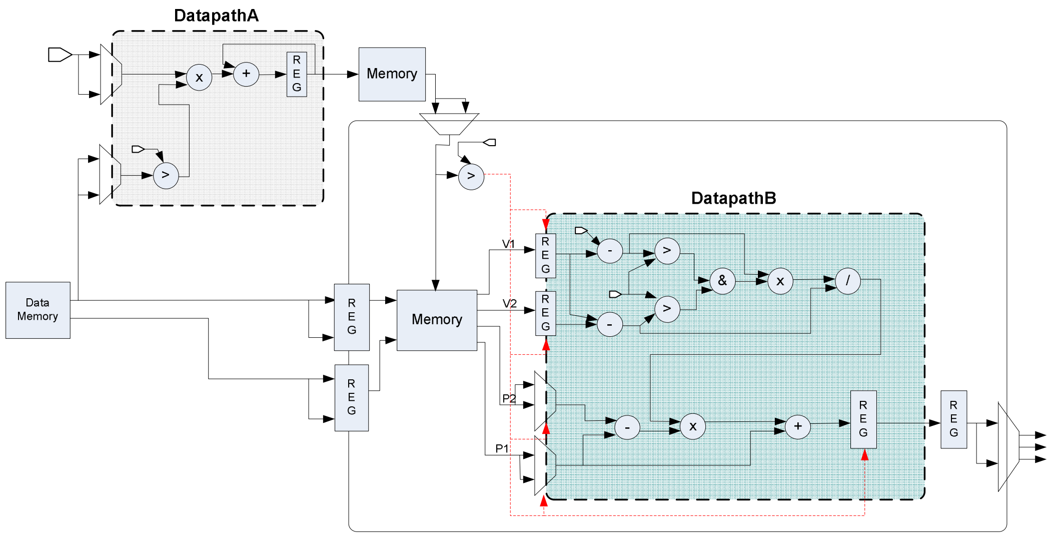


Fig. 14 Data path of the optimized implementation

This figure (Fig.14) shows two Datapaths; the first DatapathA contains computations operations (1 multiplier, 1 adder and 1 comparator) used to calculate the index and a register which is stored the intensities of eight pixels forming the voxel.

The second DatapathB contain calculations units (2 multipliers, 3 subtractors, 3 comparators, 1 adder and 1 divider) limited by the factorization frontier presented as FF3 in *GFCDD* (Fig.8), it is used to calculate the intersection points by interpolation linear. This DatapathB contains too two registers to save intensities and the pixel coordinates as input.

These two Datapaths are connected to three memory which containing the coefficients needed to implement the *Marching Cubes* algorithm.

Based on the Fig.13, we propose a **PE** (Fig.15) composed of two Datapaths containing registers backup as well as computation operations.

IV. Elementary Processor Design

According to the *GFCDD* of the *Marching Cubes* algorithm already modeled with the *SynDEx-IC* and the *SynDEx* and based on the exploration conclusions, we present the data path of the *Marching Cubes* implementation for voxel processing (Fig.14).

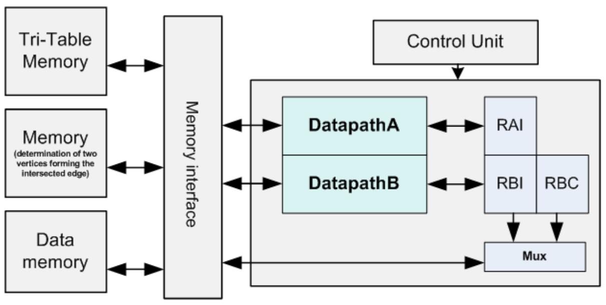


Fig. 15 The architecture of PE

This **PE** include three memory; the first one contains all possible topologies present in each cube (*Tri-table*), the second determines the two vertices forming the edge which intersected by the interested surface and the third memory contains the image data.

A memory interface is designed to ensure an efficient distribution of data to the Datapaths.

This elementary processor has been implemented and validated on two *FPGA* families, *Xilinx Virtex4* (xc4vlx200-10ff1513) and *Altera CycloneV* (5CGXC7) (Table.II).

	Slices/LEs	FlipFlop	DSP	BRAM
CycloneV(5CGXC7)	1151	1309	11	205 M10K
Virtex4(xc4vlx200)	1689	1704	2	128

Table II. Results of the PE implementation

The synthesis result presented by table.II shows that the elementary processor designed consumes on CycloneV 1151 LEs, 1309 of FF (flip-flop), 11 of DSP and 205 M10K.

On Virtex4 this core consumes 1689 Slices, 1704 of FF (flip-flop), 2 of DSP and 128 BRAM.

This result provides an elementary processor core which working at 43MHz and using only 1% of resources (Slices/LEs) *FPGA* (table2).

The obtained performances allow us to envisage a multiprocessor architecture implementation on *FPGA*.

V. Conclusion

In this paper we have studied the algorithmic and architectural exploration of the *Marching Cubes* algorithm using the *AAA* methodology and *SynDEx* tool based on the optimization constraints to define a parallel multiprocessor architecture that accommodates an optimized implementation.

By exploring the data parallelism of the algorithm, we have seen that the number of blocks to be parallelly processed is determined either by the image resolution (version1) or by the slice number (version2).

We have focused on the first version, and we have modeled this algorithm within the *SynDEx*. We have made a first implementation of an elementary processor of this architecture and defined the adequate topology for this **PE** type of and this application.

Following this work, we will focus on the second version to well optimize the algorithm by using two parallel programming models: the *MPI* and *OpenMP* models.

References:

[1] A. Lopes, K. Brodlie, Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. IEEE Transactions on Visualization and Computer Graphics, 2003, 9(1) p: 16-29.

[2] Timothy S. Newman_, Hong Yi , "A survey of the marching cubes algorithm", Elsevier, Sciencedirect, Computers & Graphics 30 (2006) 854–879.

[3] V. Gelder, J. Wilhelms, Topological considerations in isosurface generation. ACM Trans Graph 1994; 13, p: 337–75.

[4] E. Belhaire, E. Bourennane, G. Bouvier, D. Demigny, P. Garda, L. Kessal, L. Lacassagne, F. Lohier, M. Paindavoine, Y. Sorel, L. Torres, and S. Weber. Méthodes et architectures pour le traitement du signal et des images en temps réel, chapter 5: Y. Sorel, Méthodologie AAA et logiciel SynDEx, pages 79-108. IC2. Hermes, 2001.

[5] Paul Bourke, 3D Contouring, Marching Cubes, Surface Reconstruction, 1994.

[6] N. Ghezal, S. Matiatos, P. Piovesan, Y. Sorel, M. Sorine. SYNDEX, un environnement de programmation pour multiprocesseur de traitement du signal : Mécanismes de communication. Rapports de Recherche INRIA n° 1236 (1990).

[7] The AAA methodology and SynDEx. INRIA, 1999. http://www-rocq.inria.fr/syndex/

[8] C. Lavarenne, O. Seghrouchni, Y. Sorel, M. Sorine. The SynDEx software environment for real time distributed systems design and implementation. Proc. of the European Control Conference (Juillet 1991).

[9] L.Kaouane, M.Akil, Y.Sorel, T.grandpierre. A methodology to implement real-time applications onto reconfigurable circuits, Special issue on Engineering of Configurable Systems of the Journal of Supercomputing, Kluwer Academic Publisher, Vol.30, No.3, Dec. 2004 .

[10] William E. Lorensen and Harvey E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm," Computer Graphics, 21(4), 163–169, July 1987.

[11] T. Grandpierre, Y. Sorel. "From Algorithm and Architecture Specifiations to Automatic Generation of Distributed Real-Time Executives: a Seamless Flow Graphs Transformations", IN Procs. IEEE Formal Methods and Models for Codesign Conference (MEMCODE'2003), pp: 123-133, Mont Saint-Michel, France, June 24-26, 2003.

Manel MILI received the Bachelor's and Master's Degree in the National Engineering School of Sfax, University of Sfax, in 2007 and 2008 respectively.

Since 2009, she has been working as a Research Scientist at the Research team of Medical Imaging Technology (TIM), Faculty of Medicine at Monastir University of Monastir where she prepares his thesis.

She has also four published paper in international conference in the same areas. His areas of interest include dedicated architecture of medical image processing.

Bouraoui MAHMOUD received the Bachelor's, Master's Degree and doctorate in the Faculty of Science at Monastir from Mastir University, in 1997, 1999 and 2006 respectively.

Since 2008, He is currently Assistant Professor in the Department of Industrial Electronics in the National Engineering School of Sousse, University of Sousse.

He has four published papers in international journals. His areas of interest include embedded processor, embedded system, medical imaging, codesign HW/SW.

Mohamed Hedi BEDOUI Since 2011, He is currently Professor in Biophysics in the Faculty of Medecine at Monastir from Monastir University. He has many published papers in international journals. His areas of interest include Biophysics, medical imaging processing, embedded system, codesign HW/SW.