

# Parallel Architecture for the Solution of Linear Equations Systems Based on Division Free Gaussian Elimination Method Implemented in FPGA

R. MARTINEZ, D. TORRES, M. MADRIGAL, S. MAXIMOV

Program of Graduated and Investigation in Electrical Engineering as the Morelia Institute Technology

Av. Tecnológico 1500, Col. Lomas de Santiaguito, Morelia, Michoacán  
MEXICO

ruben.martinez08@cfe.gob.mx <http://elec.itmorelia.edu.mx>

**Abstract:** - This paper presents a parallel architecture for the solution of linear equations systems based on the Division Free Gaussian Elimination Method. This architecture was implemented in a Field Programmable Gate Array (FPGA). The division-free Gaussian elimination method was integrated in identical processors in a FPGA Spartan 3 of Xilinx. A top-down design was used. The proposed architecture can handle IEEE 754 single and double precision floating-point data and the architecture was implemented in 240 processors. Also, an algorithmic complexity of  $O(n^2)$  was obtained using a  $n^2$  processors scheme that perform the solution of the linear equations. Moreover, the parallel division-free Gaussian elimination method, the architecture's data distribution and the internal processor-element (PE) architecture are presented. Finally, this paper presents the obtained simulation results and synthesis of the modules designed in very high-speed integrated circuit hardware description language (VHDL) using 40 and 100 Mhz frequencies.

**Key-Words:** - Field Programmable Gate Array (FPGA), Parallel Processing, Parallel Architectures, linear systems equations, Division Free Gaussian elimination Method.

## 1 Introduction

Linear systems are commonly found in many scientific fields, and they can range from a few to millions variables. Consequently, for large linear equation systems, the system solution involves long computational times. For this reason, parallel processing emerges as a good option for the solution of linear systems [1]. In the past years, many methods for the solution of linear system have been proposed. These methods present many advantages and drawbacks and the selection of the method rests in the problem to be solved [2]. The technological development in parallel processing, computer systems and electronic digital devices have evolved engineering. The design of new parallel architectures for the solution of engineering problems is becoming popular due to the advantages of parallel processing [3]. For instance, repetitive process can be executed simultaneously in order to reduce computational efforts. Therefore, the application of parallel techniques in methods for linear systems solutions can improve significantly its performance [5]-[7].

Recently, parallel processing has been applied for the solution of problems in image processing, finite

element, mathematical algorithms and power electrical systems [9]. It is important to say that due to the behavior and mathematical modeling of some physical systems, its solution of can be parallelized [10, 11].

Section 2 presents the mathematical model of the one-step division-free Gaussian elimination method as well as its equations, conditions, algorithmic complexity and computational algorithm. The data distribution between processor elements (PEs) at each iteration is shown in Section 3. Section 4 presents the proposed parallel architecture, named here Division-Free Parallel Architecture (DFPA). It also depicts the vertical and horizontal processing performed by PEs. The implementation in FPGA of the proposed architecture, the obtained simulations and synthesis of every module of the designed architecture is presented in Section 5. Section 6, shows the performed tests, obtained results and comparatives of the behavior of the proposed architecture against other architectures reported in the literature. Finally, Section 7 presents the conclusions.

## 2 One-Step Division-Free Bareiss Method

Recently, many researches are focused in the design of parallel algorithms and parallel architectures which are square-root-free and division-free [1, 8,]. The main reason is that the division operation is time and space consuming. Moreover, the division operation presents numerical instability and a critical accumulative effect of roundoff error for large input matrices [8]-[9]. The characteristics of the one-step division-free method make it suitable for its implementation in an array of processors. Let a linear system of equations be given by.

$$\begin{aligned} A &= (a_{ij}), 1 \leq i, j \leq n, \\ b &= (a_{ij}), 1 \leq i \leq n, n+1 \leq j \leq m \\ x &= (a_{ij}), 1 \leq i \leq n, 1 \leq j \leq m-n. \end{aligned}$$

To solve  $Ax = b$ , the matrix  $A$  should be reduced to a diagonal form or a triangular form with subsequent back substitution.

In general, the algorithm that reduces the matrix  $A$  to a diagonal form is more complex than the algorithm that reduces it to a triangular form. In this paper, the one-step division-free Gaussian elimination algorithm is used to reduce the matrix  $A$  to a diagonal form. The one-step division-free Gaussian elimination algorithm is described in (1).

$$\begin{aligned} a_{ij}^{(0)} &= a_{ij}, \quad 1 \leq i \leq n, 1 \leq j \leq m; \\ 1 \leq k \leq n, 1 \leq i \leq n, k \leq j \leq m; \end{aligned} \quad (1)$$

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)}, & \text{if } i = k \\ \begin{pmatrix} a_{kk}^{(k-1)} & a_{kj}^{(k-1)} \\ a_{ik}^{(k-1)} & a_{ij}^{(k-1)} \end{pmatrix}, & \text{otherwise} \end{cases}$$

At each iteration, it is possible to compute all the  $2 \times 2$  determinants at the same time. Consequently, computation of the determinants is given by  $nd$  where  $nd$  is the number of determinants to be computed in one iteration,  $n$  is the number of rows and  $m$  is the number of columns of matrix  $A$ . Equation (2) shows the algorithm complexity.

$$nd = n \cdot m \quad (2)$$

where  $O(n)$  is the algorithmic complexity for the number of determinants to compute in each iteration,  $n$  is the number of rows,  $m$  is the number of columns, and  $m = n+1$ ; then:

$$nd = n^2 + n \quad (3)$$

Since the major exponent in (3) is  $n^2$ , the algorithmic complexity is  $O(n^2)$ . Data dependency exists in each iteration. Since iteration  $k+1$  requires the data computed in the preceding iteration, the iteration  $k$  has to be previously computed.

### 2.1 Proposed Architecture

The proposed architecture consists in an array of processors distributed in a matrix form, where the number of PEs is  $n^2$ .

A master processor is in charge of the data distribution between PEs. The master processor distributes the data that will be used for each PE during the first iteration.

The PE performs sums and multiplications to compute a  $2 \times 2$  determinant. The PEs require eight memory locations: the first four to store the four variables, the fifth and sixth location to store the multiplication results, the seventh position to store the sum and the eighth location to store the PE identification number. Each processor can be uniquely identified by its position onto the processor grid.

---

#### Algorithm: One step division free method

---

```

for (k=0; k<n; k++)
  for (i=0; i<n; i++) // Row
    for (j=0; j<n+1; j++) // Column
      if (k==i)
        D[i][j]=C[k][j];
      else
        D[i][j]=(C[k][k]*C[i][j])-(C[i][k]*C[k][j]);
      end if
    end for
  end for
for (p=0; p<n; p++) // matrix C
  for (q=0; q<n+1; q++)
    C[p][q]=D[p][q];
  end for
end for //end program

```

---

### 3 Data distribution

The first task needed to perform the algorithm is the data distribution. The data is sent to each processor; therefore the processor can perform its internal operations. The master processor sequentially sends the data to the first  $n$  processors.

The number of the executed iteration is considering as the position of the processor into the grid. Therefore, each processor computes the data according to its position. For instance, the processor  $P_{11}$  computes the element  $a_{11}$ , the processor  $P_{12}$  computes the element  $a_{12}$ , and successively every

processor  $P_{ij}$  computes its corresponding element  $a_{ij}$ . At the beginning of the algorithm, the data from the original matrix is taken as the input data for the iteration 0. Consider  $k$  as the number of iteration,  $i$  as the number of row and  $j$  as the number of column. The processors are distributed in a matrix form shown in Fig 1.

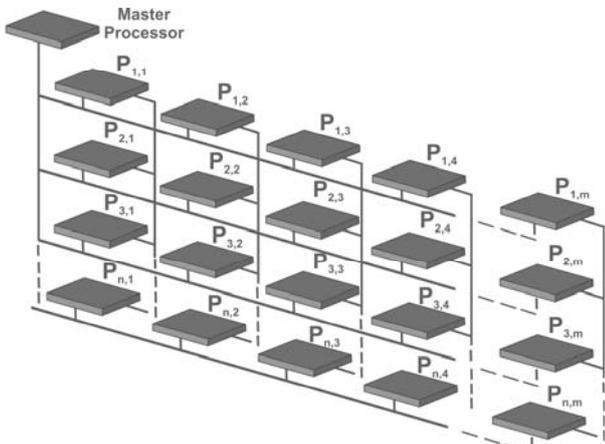


Fig. 1. Distribution of the processor in matrix form.

The data that correspond to the position is stored at each processor. For instance, the processor  $P_{11}$  store the element  $a_{11}$ , the processor  $P_{12}$  store the element  $a_{12}$ , and successively. The master processor sends the input data to every processor. Fig. 2 shows the data distribution.

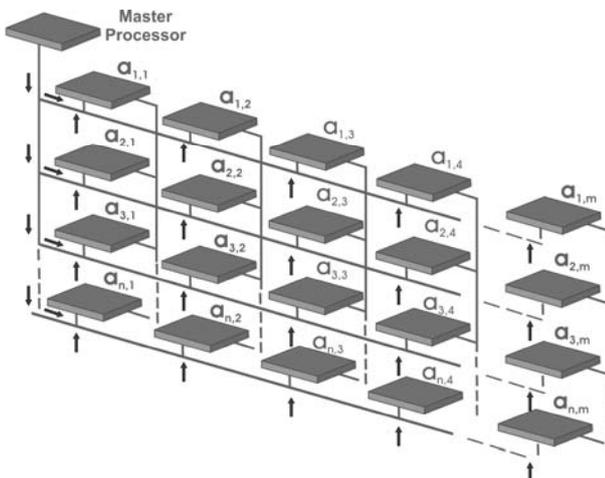


Fig. 2. Data distribution of elements  $a_{ij}$  into the processors  $P_{ij}$ .

The elements  $a_{11}, a_{12}, a_{13}, \dots, a_{1n}$  for the first iteration are exactly the same for the iteration 0. The first row of the new matrix corresponding to iteration 1 is equal to the first row of the original matrix; therefore, its computation is not required. These elements are stored in the master processor, one by one until

reach the  $n$  elements. The element  $a_{11}$  is stored in its corresponding processor  $P_{ij}$  when  $k \neq i$ . If  $k = 1$  then all the processors need the term  $a_{11}$  due to this element is used in the computation of all the determinants required for the computation of the new value in the next iteration of the algorithm. Moreover, this element is stored in all the processor in a single clock cycle. Fig. 3 shows this distribution.

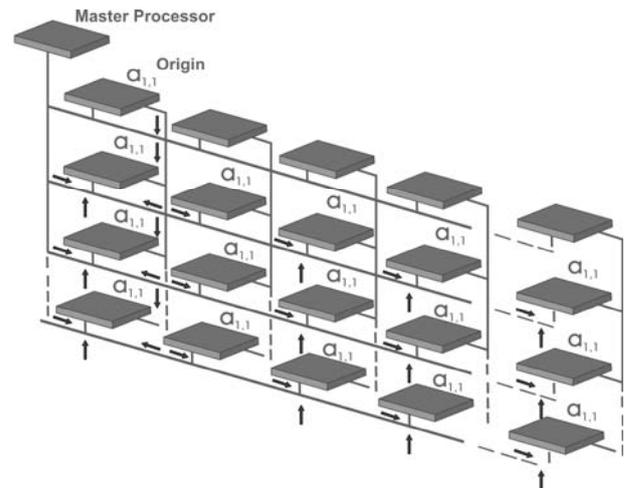


Fig. 3. Data distribution of the elements  $a_{kk}$  into the processors  $P_{ij}$ .

For the rows a special condition has to be considered. Each processor needs the element that corresponds to the row number into the matrix and the current iteration. For instance, for row 2 and iteration 1, the element  $a_{21}$  is stored in all the processor located in row 2 of the matrix of processors. This special data distribution is shown in Fig. 4.

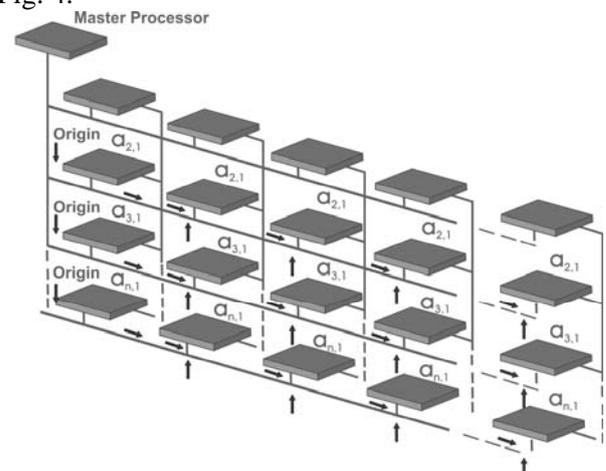


Fig. 4. Data distribution in the rows of the matrix of processors.

The columns also have a common data that is stored in all the processor located in the same column. For

example, for column 2 and iteration 1, the element  $a_{12}$  is stored in all the processors corresponding to that column number. This behavior is present in every column. Fig. 5 depicts this data distribution.

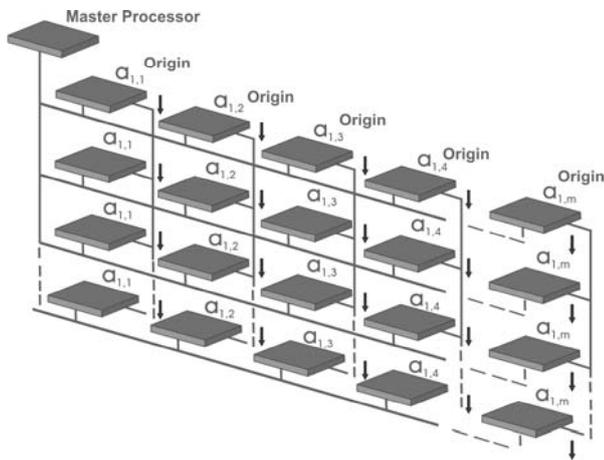


Fig. 5. Data distribution in the columns of the architecture.

The data distribution presented in this section is performed to compute the first iteration of the algorithm. The data stored in each processor is used to compute the determinants that produce the new input data required for the next iteration  $k+1$ .

### 4 Parallel Architecture of the Processor

This section presents the proposed architecture DFPA, based in the Bareiss method, for the solution of linear equations. Also, the vertical and horizontal processing into the array of PEs is described. The proposed architecture is composed by an array of PEs, each PE compute the Bareiss determinant defined in (1). The processors are arranged as a grid. Fig. 6, depicts the processor grid.

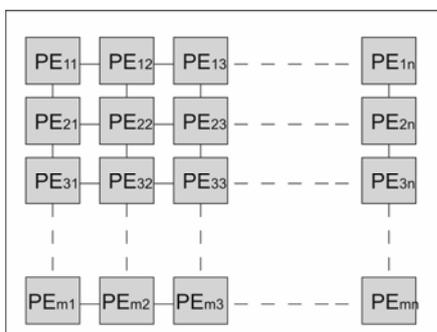


Fig. 6. Matrix form of the grid of EPs.

Every PE into the DFPA Architecture receives the

Bareiss coefficients through the rows of the grid. The processor  $PE_{1n}$  receives the coefficient  $a_{in}$  and successively for the rest of processors until the processor  $EP_{mn}$  receives the coefficient  $a_{mn}$ . These coefficients conforms the Bareiss determinant coefficients. Once the four coefficients are received in each PE the determinant is computed. This processing is performed in horizontal form as shown in Fig. 7.

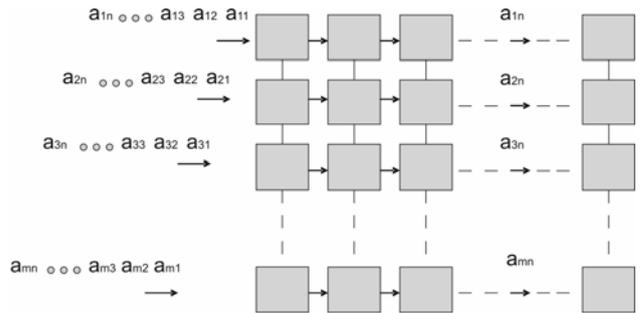


Fig. 7. Horizontal processing of the coefficients by rows.

The coefficient  $a_{11}$  is sent to every PE using the processor element  $EP_{11}$ . At first, this coefficient is sent in vertical form to every  $EP_{1n}$ , then each processor sends this coefficient in horizontal form to all the rows in the grid of PEs as shown in Fig. 8. The horizontal processing is used for the Bareiss determinant computation. After that, the result is sent to the others processors for the computation of the next iteration.

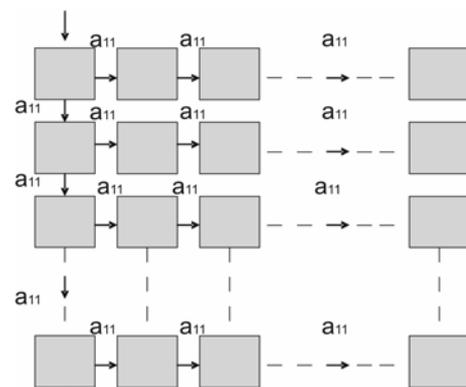


Fig. 8. Processing of element  $a_{11}$ .

Fig. 9 shows the vertical processing. Once the first determinant at each iteration is computed, the vertical processing is performed to compute and send the obtained results to the rest of processors in vertical

form. The vertical processing comprises of sending the first row ( $a_{11}, a_{12}, a_{13}, \dots, a_{1n}$ ) to the PEs arranged in the following rows on the processor grid. This procedure permits the Bareiss determinant of the next iteration to be constructed.

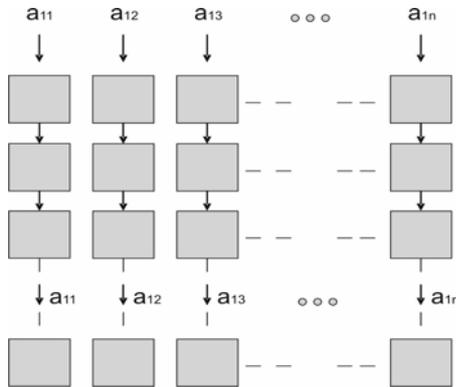


Fig. 9. Vertical processing of the coefficients in the first row of the proposed architecture.

A horizontal processing is performed to compute and send the first column coefficients ( $a_{11}, a_{21}, a_{31}, \dots, a_{m1}$ ) to the rest of the PEs arranged in subsequent columns on the grid as shown in Fig. 10. This procedure completes the Bareiss determinant for iteration 1.

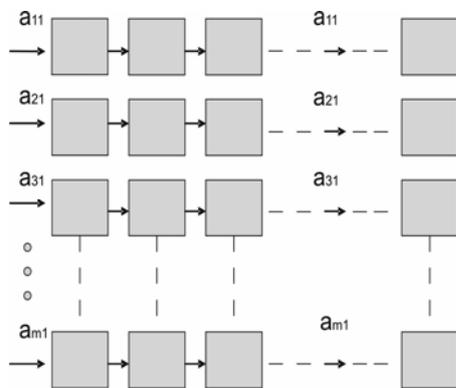


Fig. 10. Horizontal processing of the coefficients in the first column of the proposed architecture.

Fig. 11 depicts the allocation of the coefficients onto the PEs grid for any iteration. Also, it can be seen in the figure how the different determinants are conformed.

A PE basically consists of a multiplier, an adder and an accumulator. The PE receives four 32-bits data to solve a Bareiss determinant performing two multiplications and a sum. Also, each received data is stored into a register in the internal memory of the processor.

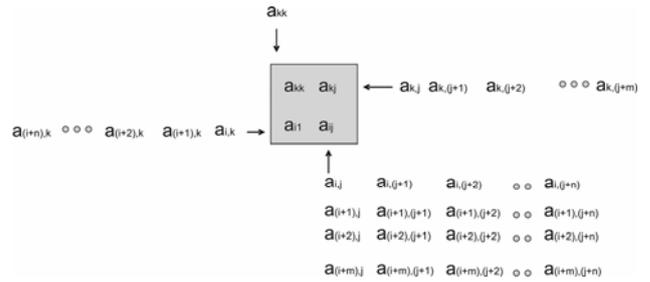


Fig. 11. EP processing for iteration  $n$ .

Similarly, the multiplication results are stored into two registers and the sum result in another register. Consequently, each PE is composed of eight registers. In order to solve the linear equation system, the PEs need to communicate with their processor neighbors. Fig. 12 shows the internal blocks that compose a PE.

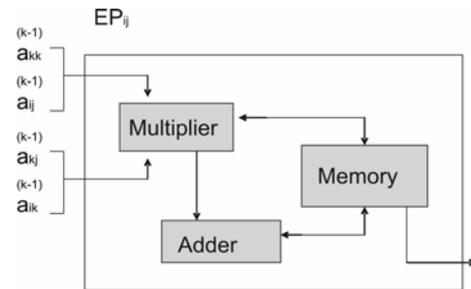


Fig. 12. Processor Element.

## 5 FPGA Implementation of the Proposed Architecture

The synthesis of the designed modules of the proposed architecture (DFPA) can handle IEEE 754 single and double precision floating-point data. The simulations were performed using the ModelSim 6.3f software and VHDL language. Also, the VHDL language includes IEEE standard libraries for arithmetic operations and data conversion functions [9].

### 5.1. VHDL Simulations

These libraries were used for the design of the modules presented in this section. The designed Processor diagram block is presented in Fig. 13. It can be observed that this processor is composed by the five modules described before.

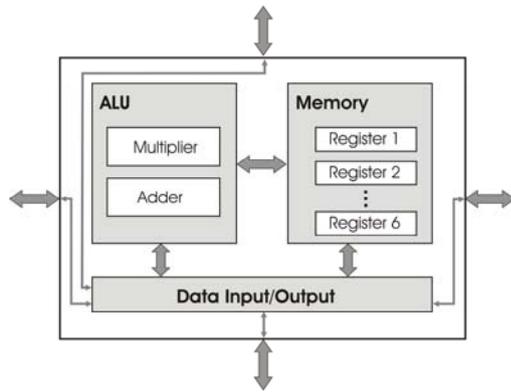


Fig. 13. Processor diagram block.

### 5.1.1. Adder/Subtractor Module

The designed Adder can sum and/or subtract positive and negative double precision floating-point numbers. In [8], a detailed description of floating-point numbers implemented in FPGAs is presented. Fig. 14(a) shows the simulation of the sum of two numbers in ModelSim 6.3f software. It can be seen in the figure the result and the time needed for the computation of the sum that, in this case, is 4,000 ns approximately.

### 5.1.2. Multiplier Module

This module computes the multiplication of two positive/negative numbers. This operation is required for the Bareiss determinant computation in the designed parallel architecture. This module requires two 32-bits single precision floating-point numbers to obtain a 64-bits double precision floating-point number as result. Fig. 14(b) shows the simulation of a 32-bits multiplication, its result and the execution time (1,500 ns).

### 5.1.3 Memory Register Module

The memory module is essential for the storage of the coefficients of the Bareiss determinant, the sum result and the multiplication result during the determinant computation. This module consists of eight 64-bits registers and is basically the processor memory of each PE.

Fig. 14(c) shows the simulation of this module where a data is stored in a memory register in 1,325 ns.

### 5.1.4 Communication Module

This module was designed for serial data transmission between processors. The PEs send and receive 64-bits data to perform mathematical

operations. Also, the master processor sends an identification number to every PE. Fig. 14(d) shows that the transmission time for a 64-bits data is 6,300 ns.

### 5.1.5 Control Module

This module manages the data transmission between processors. Each PE uses this module for the identification of neighbors to sending their results. The time needed for sending a control data is 100 ns, as shown in Fig. 14(e).

Table 1, shows the total execution time for the  $2 \times 2$  determinant computations per processor. The result is obtained in 21.7  $\mu$ s. Since all determinants are computed simultaneously, the time needed for the execution of the Bareiss algorithm per iteration is 21.7  $\mu$ s multiplied by  $n$ .

Table 1  
Times Measurements.

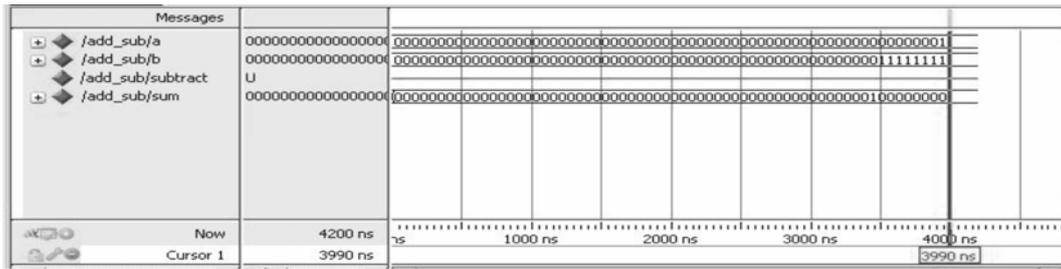
Description	Operations	Time per operation (sec)	Total time per operation (sec)
Adder-subtractor	1	$4000 \times 10^{-9}$	$4000 \times 10^{-9}$
Multiplier	2	$1500 \times 10^{-9}$	$3000 \times 10^{-9}$
Memory registers	6	$1500 \times 10^{-9}$	$9000 \times 10^{-9}$
Data input/output	4	$1325 \times 10^{-9}$	$5300 \times 10^{-9}$
Control	4	$100 \times 10^{-9}$	$400 \times 10^{-9}$
Total Time			$21700 \times 10^{-9}$

### 5.2. FPGA implementation

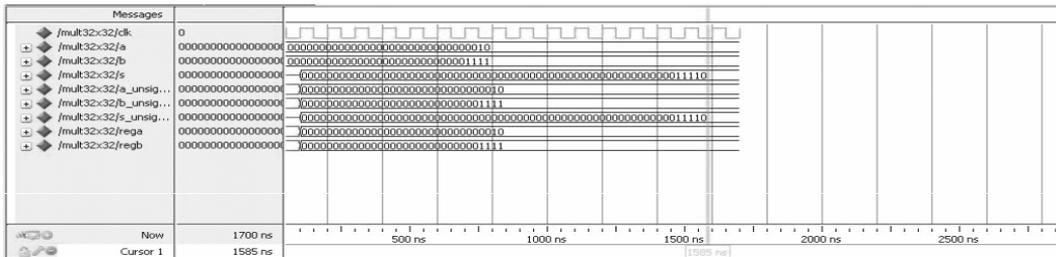
A FPGA device, also named LCA (Logic Cell Array) is used for the processing phase. The FPGA's consist in a bidimensional matrix composed of configurable block that can be connected by general resources of interconnection [7, 9].

The synthesis of the architecture was developed in the Xilinx ISE 8.1i software for FPGAs of the Spartan 3 family of Xilinx, to obtain the number of Gates, IOBs, CLBs, Slices, Shift, Flip Flops, and LUTs. The connection diagrams between digital components with different abstraction levels were also obtained.

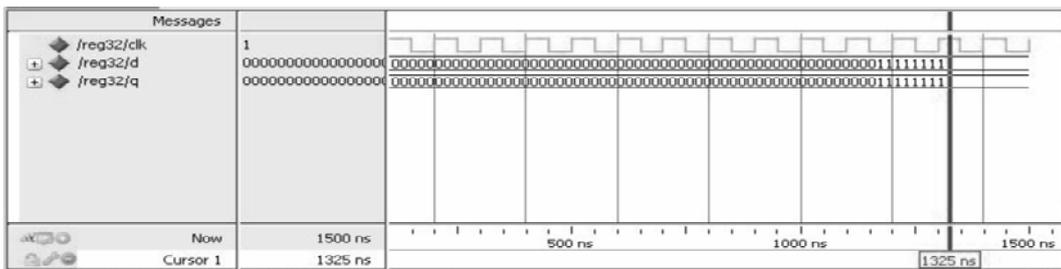
In this section, the synthesis of the modules of proposed parallel architecture, based in the parallelization of the one-step division free Gaussian elimination, is presented. The modules were designed in VHDL language and simulated using the ModelSim 6.3f software.



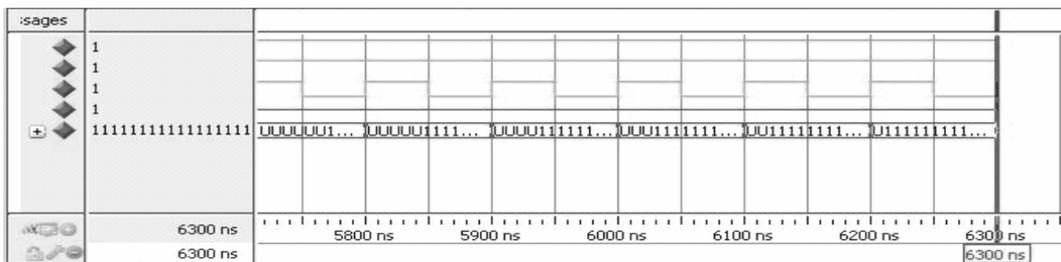
(a)



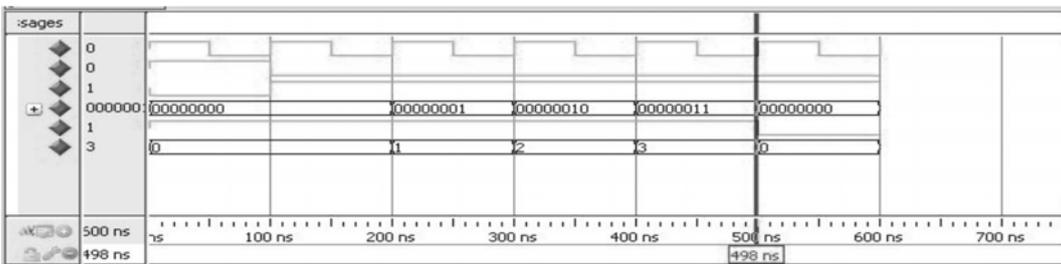
(b)



(c)



(d)



(e)

Fig. 14. (a) Adder/Subtractor simulation. (b) Multiplier simulation. (c) Data storage simulation. (d) Data transmission simulation. (e) Control module simulation.

The proposed architecture is composed by the following components or modules:

- An adder-subtractor.
- A multiplier.
- A serial data input/output
- Memory registers
- Control

Every processor has an ALU composed by an adder-subtractor of 64 bits and a multiplier of 32 bits. Moreover, every processor contains a memory of 6 registers of 64 bits, four registers to store the four elements needed to compute the 2x2 determinant, one register to store the resulting determinant and the final register stores the processor number. The data of every processor is sent sequentially. This task is performed by a component inside each processor. The master processor manages the data distribution between the processors by using a counter. All the modules were programmed in VHDL language and it includes standard IEEE libraries. In the simulations of the designed VHDL modules presented in this paper, a clock period of 100 ns was considered.

### 5.2.1. Adder-subtractor

The adder and the subtractor contained in the ALU of the processor are used for the computation of the determinant in (1). The adder is able to add and subtract negative and positive numbers in a range of 0 to  $2^{64}$ . Internally, this block is composed by an adder, a subtractor, a multiplexor, flip flops and logic stages. Fig. 15(a) depicts the formed block obtained from the synthesis of this module. It can be observed the inputs and outputs of the module.

### 5.2.2. Multiplier

The designed multiplier is able to multiply two positive or negative numbers. This module is important to the computation of the determinant. The multiplicand and the multiplier numbers can be 32 bits number or a decimal number in the range of 0 to  $2^{32}$ , the result is a 64 bits number. This module is composed by flip flops type D, multipliers, CLB's, and IO's. The block corresponding to this module is shown in Fig. 15(b).

### 5.2.3. Memory registers of 64 bits

The memory registers of 64 bits are used for data storage. Every processor has eight registers that compose its memory. Only one register is fixed and it contains the processor number. The other registers store the data required in the computation of the determinant given in (1). The block of this module is shown in Fig. 15(c).

### 5.2.4 Serial data input/output

This module performs the sending and reception of data between processors. This module sends and receives information in serial form. Data of 64 bits is sent and received for mathematical operations. Also, the processor number, that identifies each processor, is sent using this module. Fig. 15(d) shows the resulting block of the data input/output module.

### 5.2.5 Data sending controller

The master processor manages the data sending to every processor. First, the master processor sends a number of identification, called processor number, to every processor. Once all the processors have been identified, the master processor can send the elements required for the computation of the determinant. The data sending controller module was designed to identify each processor. The block of this module is shown in Fig. 15(e). The synthesis of the modules describes in this section are shown in Table 2.

Table 2  
Number of components per Processor

Module	Total Gates	CLB's	IOB's	Shift	Mult 18x18	FF	Lut 4 input
Adder-subtractor	765	32	193			64	64
Multiplier	17829	56	129		4	64	111
Memory registers	515		129			64	
Data input/output	267	2	4	4		1	
Control	756	42	12			33	47
Total Components	20132	132	467	4	4	226	222

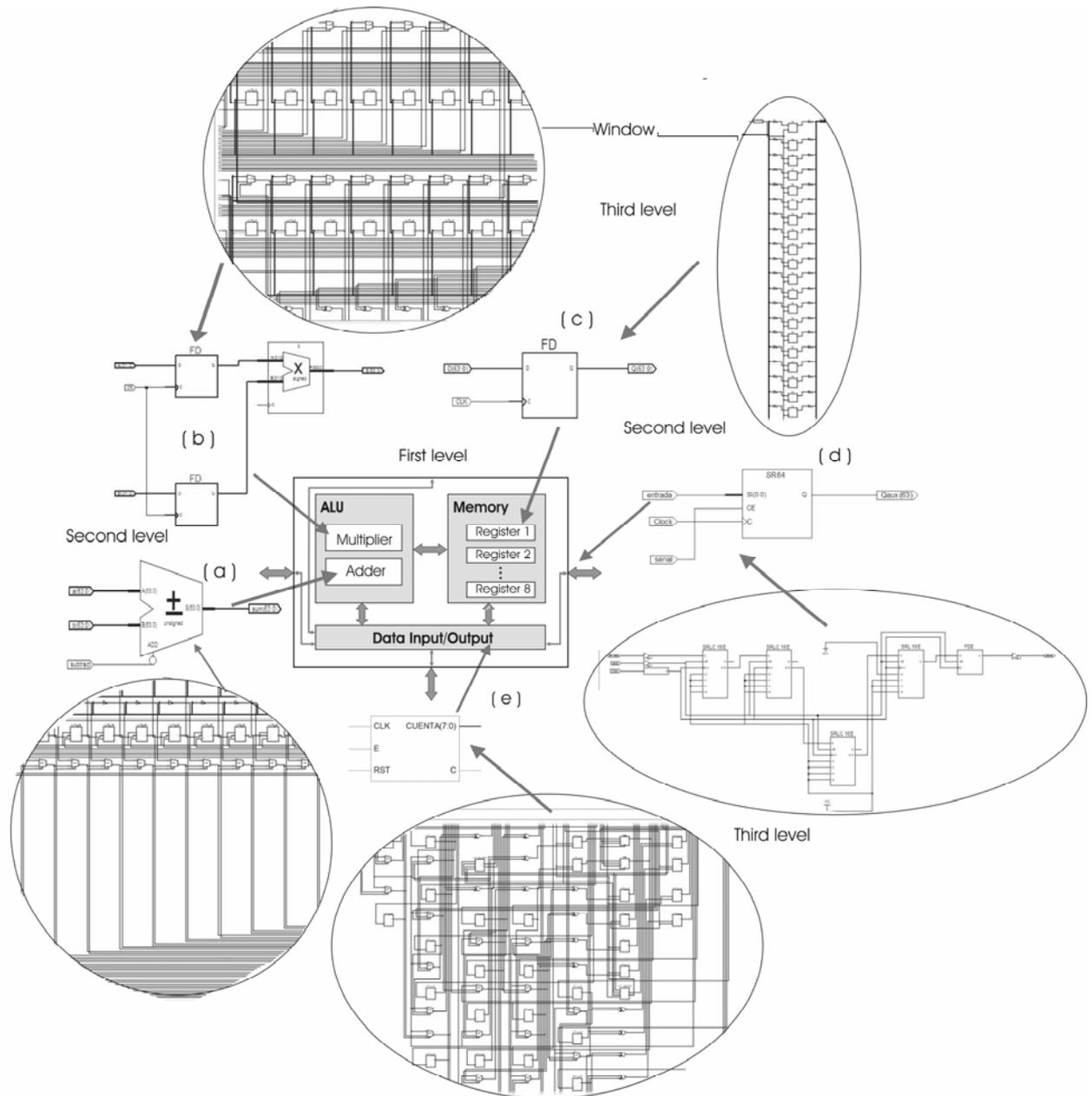


Fig. 15. (a) Adder-subtractor block, (b) Multiplier block, (c) Memory register block, (d) Data input/output block, (e) Data sending controller block.

## 6 Experimental results

In this paper, a parallel architecture, named here Division-Free Parallel Architecture (DFPA), was designed. A comparison of the obtained results against the results reported in [12] is presented. In [12], a parallel architecture for linear equation solution using the LU algorithm, named here “Parallel LU” is described. This architecture uses a 40 Mhz frequency for different matrix sizes.

Table 3 shows a comparison of the obtained times

of the DFPA against Parallel LU. An improvement factor is also presented. If this factor is greater than one, then the DFPA is better than the Parallel LU.

Table 3  
Comparison DFPA vs Parallel LU

Matrix	DFPA (sec)	Parallel LU (sec)	Improve
24	1.36E-04	6.84E-04	5.02
30	1.70E-04	1.41E-03	8.26
36	2.04E-04	1.17E-03	5.73
42	2.38E-04	1.66E-03	6.96
48	2.72E-04	3.00E-03	11.02
54	3.06E-04	4.89E-03	15.94
96	5.45E-04	1.67E-02	30.58

Fig. 16 shows a plot of the results presented in Table 3, where DFPA shows a better performance than Parallel LU.

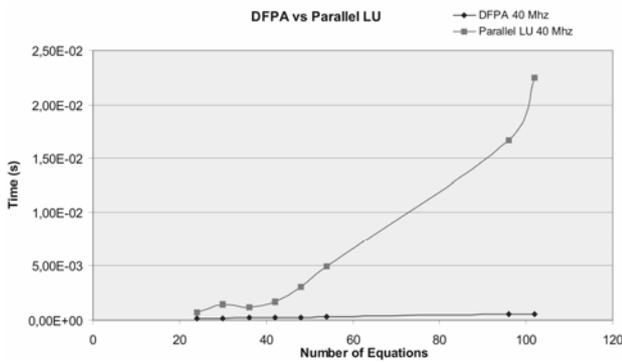


Fig. 16. Comparison DFPA vs parallel LU

Also, Fig. 17 shows the improvement factor of the DFPA against the architectures presented in [12]. It can be seen in the figure that DFPA is faster than the Parallel LU architecture.

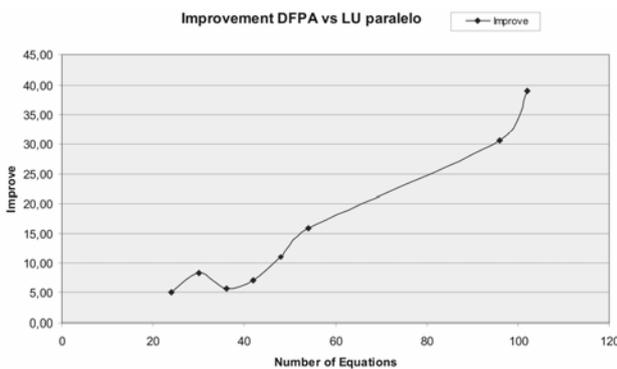


Fig. 17. Improvement DFPA vs parallel LU

Moreover, the architecture presented in [13], named here “Pipeline LU”, was compared against the proposed architecture. Also, a comparison with the architecture Pipeline LU against a 1.6 Ghz Pentium M processor is presented in [13] and used in this paper for comparison purposes. Table 4 presents the comparison of computation time for matrices between 100 and 1000 equations. It can be seen in the table that the DFPA and Pipeline LU architectures use a 100 Mhz frequency.

The comparison of the results presented in Table 4 is depicted in Fig. 18. The graph clearly illustrates that the proposed DFPA architecture is faster than the Pipeline LU in the computation of the system solution.

Table 4  
Comparison DFPA vs Pipelien LU (time ms)

Matrix	DFPA 100 Mhz	Pipeline LU 100 Mhz	Pentium M (1.6 GHz)	Improve vs Pentium	Improve vs Pipeline LU
100	0.227	0.46	9.11	40.13	2.03
300	0.681	8.76	134.20	197.06	12.86
500	1.14	40.50	661.00	579.82	35.53
800	1.82	167.60	2984.50	1639.84	92.09
1000	2.27	328.40	7871.50	3467.62	144.67

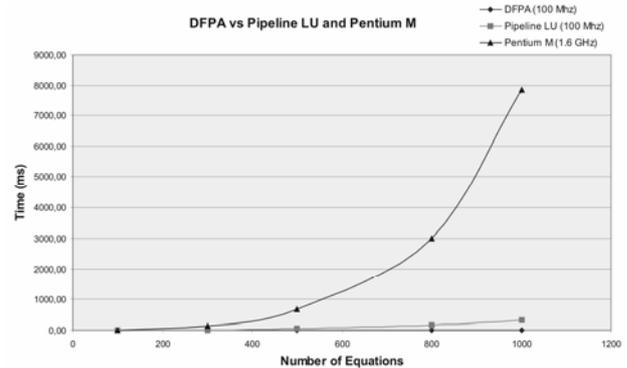


Fig. 18. Comparison DFPA vs pipeline LU and Pentium M

Also, Fig. 19 shows the improvement factor of the DFPA against the architectures presented in [10]. It can be seen in the figure that DFPA is faster than the Pentium M processor and the Pipeline LU architecture. In fact, it can be seen in Fig. 18 that for the solution of a system with 1000 equations, DFPA is 3,500 times faster than the Pentium M processor and 150 times faster than Pipeline LU approximately.

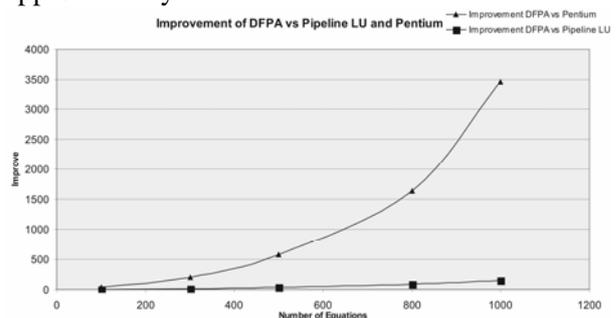


Fig. 19. Improvenmet DFPA vs pipeline LU and Pentium M

## 7 Conclusion

During the revision of mathematical algorithms for the solution of linear equation systems, the methods that use division require a major processing time and its implementation in hardware produces

complex architectures. However, the division free method proposed by Bareiss [8] presented many advantages for parallelization. For this reason, this method was selected and it is the base of the proposed parallel architecture in a FPGA.

The parallelization of the division free Gaussian elimination methods produces a simple independent process that can be implemented in identical processors and its hardware implementation is easily constructed by using basic algebraic operations. The obtained algorithmic complexity is  $O(n^2)$  under a scheme of  $n^2$  processors that solve a linear equation system of  $n$  order.

The performed simulations of the modules that compose a processor show a low time results in nano-seconds for this kind of computations.

Finally, the construction of VHDL modules for digital systems and its simulation were developed by using the ModelSim 6.3f software, whereas the Xilinx ISE 8.1i software was used for the synthesis of the matricial processor.

#### References

- [1] Shietung Peng, Stanislav Sedukhin, Parallel Algorithm and Architecture for Two-step Division-free Gaussian Elimination, *IEEE Transactions*, 0-7803-4229-1/97, 1997, pp. 489-502.
- [2] Fernando Pardo Carpio, *Arquitecturas Avanzadas*, Universidad de Valencia, España, Enero 2002.
- [3] M. J. Beauchamp, Scott Hauck, Keith D. Underwood and Scott Hemment, Architectural Modifications to Enhance the Floating-Point Performance of FPGA's, *IEEE Transactions on VLSI Systems*, Vol. 16 No. 2, February 2008, pp. 177-187.
- [4] D. Torres, Herve Mathias, Hassan Rabah, and Serge Weber, A new concept of a mono-dimensional SIMD/MIMD parallel architecture based in a Content Addressable Memory, *WSEAS, Transactions on Systems*, Issue 4, Volume 3, p. 1757-1762 ISSN 1109-2777, 2004.
- [5] Rubén Martínez Alonso, Domingo Torres Lucio, Paralelización del Algoritmo del Método de Bareiss Libre de División para Solución de Sistemas de Ecuaciones Lineales de Ingeniería Eléctrica, *4th International Congress and 2nd National Congress of Numerical Methods in Engineering and Applied Sciences* ISBN 978-84-96736-08-5, Morelia, Michoacán, México, 17-19, Enero 2007.
- [6] Ronald Scrofano, Ling Zhuo, Vicktor K. Prasanna, Area-Efficient Arithmetic Expression Evaluation Using deeply Pipelined Floating-Point Cores, *IEEE Transactions on VLSI Systems*, Vol. 16, No. 2, February 2008, pp. 167-176.
- [7] Xilinx, DS099. Spartan-3 Family, complete data sheet, <http://www.xilinx.com>, product specification.
- [8] Bareiss E. H. Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination, *Mathematics of Computation*, 22, 1968, pp. 565-578.
- [9] R. Martínez, D. Torres, M. Madrigal, S. Maximov, Parallel Processors Architecture in FPGA for the Solution of Linear Equations Systems, *8th WSEAS, Int. Conf. on System Science and Simulation in Engineering (ICOSSSE '09)*, October 2009.
- [10] Jeng-Kuang Hwang, Yuan-Ping Li, Modular design and implementation of FPGA-based tap-selective maximum-likelihood channel estimator, *WSEAS, Transactions on Signal Processing*, v.4 n.12, p. 667-676, December 2008.
- [11] K. Tanigawa, T. Hironaka, M. Maeda, T. Sueyoshi, K. Aoyama, T. Koide and H.J. Mattausch, Performance Evaluation of Superscalar Processor with Multi-Bank Register File and an Implementation Result, *WSEAS, Transactions on Computer*, Issue 9, Vol. 5, 1993-2000 (2006).
- [12] Xiaofang Wang and Sotirios G. Ziavras, Parallel Direct Solution of Linear Equations on FPGA-Based Machines, *IEEE Proceedings Symposium, (IPDPS 2003) Parallel and distributed Processing*, 2003.
- [13] Vikash Daga, Gokul Govindu, Viktor Prasanna, Efficient Floating-Point based Block LU Decomposition on FPGAs, *ERSA 2005*, pp.137-148. Las Vegas Nevada, USA, pp. 137-148, June, 21-24. 2004.