Lecture Notes in Computer Science, v. 4419, p. 326-336, 2007

Reconfigurable Parallel Architecture for Genetic Algorithms: Application to the Synthesis of Digital Circuits

Edson P. Ferlin[1,2], Heitor S. Lopes[2], Carlos R. Erig Lima[2], and Ederson Cichaczewski[1]

[1] Computer Engineering Department, Positivo University Center (UnicenP), R. Pedro V. P. Souza, 5300, 81280-230 Curitiba (PR), Brazil
ferlin@unicenp.edu.br,
[2] Bioinformatics Laboratory, Federal University of Technology Paraná (UTFPR), Av. 7 de setembro, 3165, 80230-901 Curitiba (PR) , Brazil
hslopes@pesquisador.cnpq.br,erig@utfpr.edu.br

Abstract. This work presents a proposal and implementation of a reconfigurable parallel architecture, using Genetic Algorithms and applied to synthesis of combinational digital circuits. This reconfigurable parallel architecture uses concepts of computer architecture and parallel processing to obtain a scalable performance. It is developed in VHDL and implemented totally in hardware using FPGA devices. The concept of reconfigurable and parallel architecture enables an easy hardware adaptation to different project requirements. This approach allows applies with flexibility different strategies to synthesis of combinational digital circuits problem.

1 Introduction

Recently, we have witnessed a pronounced growth the hardware and software technologies for embedded systems, with many technological options coming up every year. The use of open and reconfigurable structures is becoming attractive, especially due to its robustness and flexibility. The possibility of massive parallel processing makes reconfigurable computing a suitable technology to be applied to the growing computational demand of scientific computation.

Reconfigurable architectures or reconfigurable computational systems architecture are those where the logic blocks can be reconfigured, in their logical functions and internal functionality. The interconnection between these logic blocks, that usually performs computational tasks such as processing, storing, communication or data in/out, can be reconfigured too [6].

Because these logic blocks are implemented directly in hardware, it is possible to run algorithms exploiting the inherent parallelism of a hardware solution, achieving a throughput, much higher than if they were run in a sequential processor, subjected to the Von Neumann model [8].

For several complex problems, the solution using traditional software approach, where the hardware executes sequential algorithms, does not satisfy

the timing and performance requirements. This justifies the search for new approaches to minimize these bottlenecks. For example, reconfigurable logic allows a dramatic minimization of the processing time, when compared with a traditional software approach. Such performance is possible thanks to the parallel processing and reduced computation time, inherent to a FPGA (Field-Programmable Gate Array) implementation [9].

The motivation for using parallel processing is increasing the computational power of a limited processor. This is accomplished by exploiting the simultaneous events in a software execution [3].

The Genetic Algorithm (GA) was proposed in the 60's by Holland [7], with the initial aim of studying the phenomenon related to species adaptation and natural selection that occurs in the nature. Later, GAs becomes a powerful computational method to solve optimization and machine learning problems in the areas of engineering and computer science [4]. GAs are intrinsically parallel algorithms and are particularly attractive for parallel implementations [1].

This work proposes the implementation of a reconfigurable parallel architecture, applied to the synthesis of a combinational digital circuit by using a Genetic Algorithm. This system uses concepts of parallel processing and reconfigurable computing to achieve a scalable performance. It is developed in VHDL (VHSIC Hardware Description Language) [13] and fully implemented in hardware using a FPGA device.

2 Problem Description

The problem treated in this work consists in obtaining a minimal Boolean equation for a combinational digital circuit that implements a determined function specified by a truth-table.

In the project of logic circuits we can use many criteria to define the minimal cost expression. The complexity of a logic circuit is a function of the number and complexity of circuit gates. The complexity of a gate is, in general, a function of the number of gate inputs. Since logic circuits will implement a Boolean function in hardware, reducing the number of function literals, will reduce the number of inputs for each gate and the number of gates in the circuit. Consequently, the overall complexity of the circuit will be reduced.

The implementation described here can be used for circuits with up to four input bits and one output, such as the one shown in the example presented in table 1. In the case of problems that have two or more outputs, each output is treated independently.

This work presents a multi-objective GA for the synthesis of combinational digital circuits. The basic idea is to evaluate fitness in two stages. Firstly, to obtain a circuit that matches the truth-table, that is, an equivalent circuit. When this objective is achieved other fitness function is used. This second function aims at minimizing the amount of logic gates in the circuit.

Table 1. Example of truth-table: A, B, C and D are inputs and S is the output.

A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
S	1	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	1

3 A Reconfigurable Parallel Architecture for GAs

One of the first parallel architectures for GAs was SPGA (Splash 2 Parallel Genetic Algorithm) [5]. This architecture was applied to the travelling salesman problem and was composed by 16 genetic processors which, in turn, included the following modules: selection, crossover, fitness & mutation and statistics. Such modules were grouped using an insular parallel model. The main difference of SPGA architecture with the one proposed in this work is that the fitness function is replicated to be processed in parallel, keeping a single population.

The proposed architecture uses the computational model known as Cartesian Genetic Programming (CGP) [11]. This model, shown in figure 1, is based on a geometric set of $n \times m$ Logic Cells (LC), with n inputs and m outputs. This model was used because it is suited to the hardware parallelism given by the simultaneous configuration of the logic cells by the chromosome, and also, the way each logical function is mapped in a cell. The number of inputs (n) depends on the problem, while the number of outputs (m) depends exclusively on the number of logic cell lines of the geometric set (see figure 1).

The logical functions and the inputs for such logical functions can be selected individually in each logic cell. The connectivity of the logic cells is arbitrary, since they can use either the regular inputs of the system or the outputs of other logic cells.

This same model was used in other projects such as [2], [14], [16], thus suggesting its usefulness to this work.

This architecture is directed to the execution in the Farm model of parallelism, also known as Global Parallel GA [15]. In this model, the Master-Slave, in which the Master (Control Unit - CU) is responsible for the generation of the initial population, and the selection, crossover and mutation of chromosomes, while other Slaves (here called Processing Elements - PEs) calculate the fitness of each chromosome. This model exploits the parallelism inherent to the evaluation of fitness of individuals in the population. Each chromosome of population is treated individually and, therefore, many chromosomes can be processed simultaneously, reducing the processing time.

The architecture proposed uses a single Control Unit, and several PEs. Figure 2 presents a block diagram showing the CU and several PEs implemented using reconfigurable logic (FPGA). The host is responsible for sending to the parallel machine the configuration parameters, such as probability of mutation, crossover, number of generations, and the truth-table. After this configuration the chromosomes are sent to PEs by the CU. The PEs evaluates the chromosomes and computes the fitness. This result is send back to the CU.

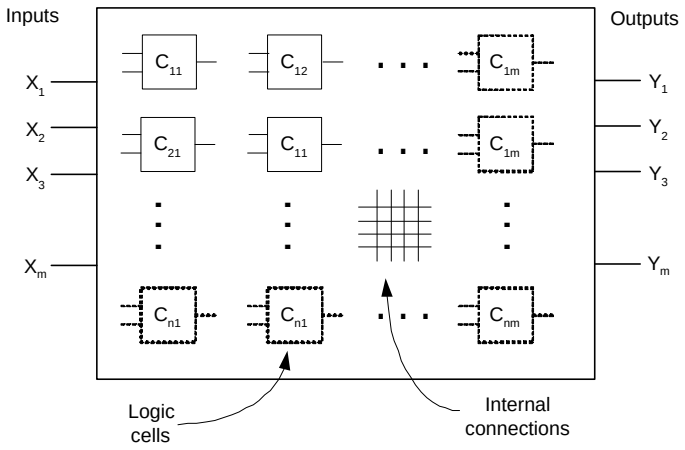

Fig. 1. Geometric mapping of logic cells in the CGP model.

The main contribution of this architecture is that it can be applied to several different problems, providing the architecture (in special, the PEs) is reconfigured for a particular problem. Besides being reconfigurable, the architecture is parallel, thus exploring an important feature of implementations in FPGAs.

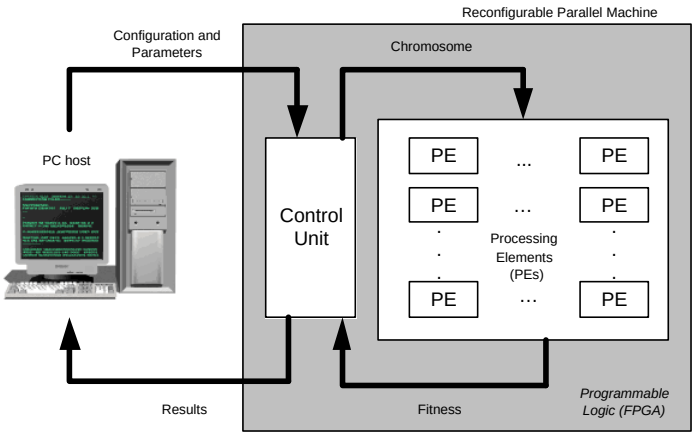

Fig. 2. General vision of the reconfigurable parallel architecture for GA.

4 Implementation

The chromosome was encoded with 25 genes (one gene for each LC) and each gene has 7 bits. Therefore, the chromosome was 175 bits-long. Each gene is responsible for the configuration of an LC, and it is composed by three fields:

address A, address B and the function. Four bits of the gene are used for the selection of the LC inputs, and 3 bits are used for selecting the function (8 possible functions).

The fitness function for this problem is multi-objective. Initially, it is evaluated how many lines of truth-table matches ("matching"). Next, the amount of null logic cells is counted ("nulls"). The fitness value is composed by the concatenation of three information: matching (5 bits), nulls (5 bits) and output (3 bits) - number of outputs of the LC matrix that produced these results.

The parallel architecture proposed is shown in figure 3. It receives from the host four parameters: P_c (crossover probability), P_m (mutation probability), number of generations and reference truth-table. Crossover and mutation probabilities are encoded with 10 bits, representing values from 0 to 0.999. Cyclically, the system sends back to the host the best chromosome and respective fitness at each generation.

The initial population is randomly generated with 100 individuals and stored in chromosome memory. After, the individuals are sent to the PEs for computing the fitness. When individuals are under evaluation, they are kept in the scratch memory. At the same time, the best individual is updated to be sent back to the host at the end of the generation. When all individuals are processed, the CU performs the selection procedure and applies crossover and mutation operators (according to P_c and P_m). When this step is concluded, the new individuals of the next generation are stored again in the chromosome memory, and the whole process is repeated until reaching the predefined maximum number of generations.

4.1 Control Unit

The Control Unit is responsible for managing input and output data as well as the other operations previously mentioned. The width of the parallel busses connecting the CU to the PEs is proportional to the number of PEs. This makes possible the simultaneous communication between the memories (in the CU) and the PEs, eliminating a possible communication bottleneck. The CU is composed by the following components:

Random Number Generator : generates random numbers that will be used for many functions in the architecture, for instance, generation of the initial population. In this case, each individual (chromosome) is obtained by concatenation of many binary sequences (numbers). The random number generator is very important to the performance of a GA system. In this case, it was implemented in VHDL based on the Mersenne-Twister algorithm described by [10]. The implemented generator uses 16-bits numbers, allowing the economy of logic elements. This representation results in a repetition cycle similar to the conventional "rand" function of C programming language.

Chromosome Memory : stores the individuals (chromosomes) to be processed. It is composed by 100 words of 175 bits. It is a double-access memory controlled by the state machine. In a first step, this memory is used for storing

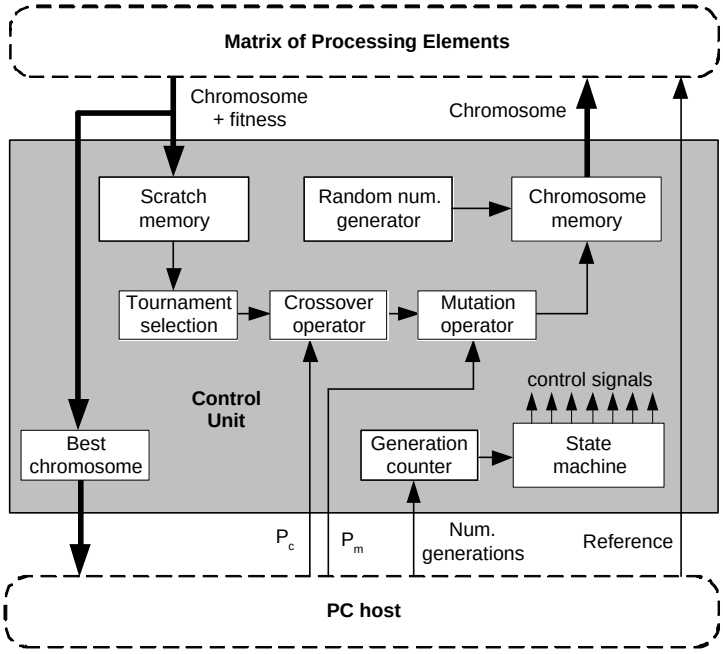

Fig. 3. Block diagram of control unit (CU).

the individuals of the initial population. Later, it stores the individuals of the forthcoming generations.

Scratch Memory : stores the already-processed individuals and their fitness value. It is capable of storing up to 100 individuals, each one composed by the chromosome (175 bits) and the corresponding fitness (13 bits). This is also a double-access memory controlled by the state machine. Selection, crossover and mutation blocks use the information stored in this memory.

Best Chromosome : finds the best individual (chromosome) of a generation, based in the fitness value. This individual is sent out to the host at the end of the generation. The best individual is obtained by comparison. For each new individual processed, this block compares its fitness with the best fitness stored.

Tournament Selection : implements the well-known stochastic tournament selection using two randomly chosen individuals of the scratch memory. The individual with highest fitness of the two is selected to be submitted to the genetic operators.

Crossover Operator : executes a classical one-point crossover operation with probability P_c. The crossover point is static and predefined to be after the 87^{th} bit. Bits 1 to 87 of the first chromosome are concatenated with bits 88 to 175 of the second chromosome to form the first offspring. The second offspring is formed in a similar way using the complimentary parts of the original chromosomes.

Mutation Operator : executes a point-mutation operation: a randomly chosen bit in the chromosome is complemented, with probability P_m.

Generation Counter : counts the number of generations. It is responsible for controlling the number of generations. It signals to other blocks of the architecture the end of processing when the maximum number of generations is reached. This counter has 10 bits and thus allows the GA to run for up to 1023 generations.

State Machine : generates all the activation signals to the other components of the architecture. The whole operational control of the system is accomplished with 20 states.

4.2 Processing Element

The internal complexity of the PEs depends on the specific application. In our case, each PE is composed by a 5×5 LC matrix (25 LCs), the truth-table and a counter to produce the input binary sequences (0 to Fh), besides the circuitry for computing the fitness (see figure 4). These elements are detailed below.

It is important to notice that the LCs matrix is dynamically reconfigured at running time. That is, for each new chromosome to be evaluated, the matrix is reconfigured, both the specific function of the LCs and the connectivity between them.

Therefore, each PE has to be as simple as possible to allow the implementation of a large number of PEs running in parallel in a FPGA device.

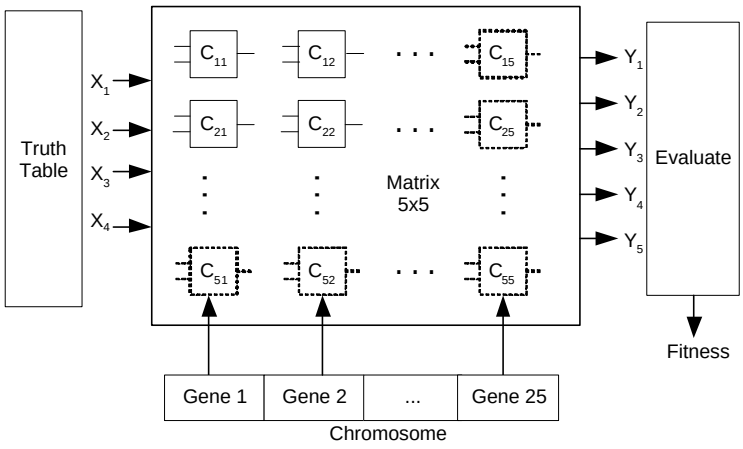

Fig. 4. Block diagram of processing element (PE).

The Processing Element is composed by following components:

Logic Cells : the LCs performs the Boolean operations shown in table 2.

Truth-Table : a 4-bits counter generates all possible input combinations (0000 to 1111) to the truth-table. Based on these values, the LCs matrix (after being configured) generates five simultaneous outputs, one for each logic expression, later used to compose the fitness value of the individual.

Evaluate : first, this block evaluates how many lines of the truth-table was satisfied (matching). After, it counts the number of "null" logic cells. Finally, it finds the number of outputs of the LC matrix that produced these results. All these information compose the fitness value.

Table 2. Logic functions of logic cells.

#	Operation	Function
0	AND	$x.y$
1	OR	$x + y$
2	XOR	$x \otimes y$
3	NOT(x)	$\bar{x}$
4	NOT(y)	$\bar{y}$
5	Wire x	x
6	Wire y	y
7	Null	null

5 Results

The hardware platform used in the development is Altera^{TM} EP1S10F484C5 Stratix device (*http://www.altera.com*). For compiling the circuitry described in VHDL was used the Quartus II Altera^{TM} design toll, version 5.0. The demands of logic elements and internal memory up as function of the number of PEs. One PE demands 5,174 logic elements and 36,300 bits of memory, occupying 48% and 3% of available resources, respectively. Table 3 shows the main components of the implemented architecture with one PE.

Table 3. Main components of the GA architecture and the respective cost of logic elements (LE)

Component	Number of LE	%
CU - RNG	631	12
CU - Selection	567	11
CU - Crossover	626	12
CU - Mutation	864	17
CU - Other	1,861	36
PE - Fitness	625	12
Total	5,174	100

The architecture was run at 50MHz clock. The first generation processing time is approximately 465μs, with the generation of initial population spending 53μs to be processed. Thus, considering that the mean time of each generation is approximately 412μs, this architecture can process around 2,426 generations (with 100 individuals) in 1 second. The fitness processing time for a chromosome in the PE is approximately 3.84μs

The evaluation of the chromosome demands 384μs. This time corresponds to 93% of the time of processing. Thus, just 28μs (7% of the time). This step that is executed sequentially, and divided as follow: 2% for Selection, 1% for Crossover and 4% for Mutation. So, the initial assumption used to justify the adopted model (Master-Slave), where the fitness evaluation of the chromosome is done by PE is correct. Thus, the parallelization of this stage, with the utilization of several parallel PEs, can reduce the total processing time for obtain a generation, what is the primary objective of this architecture.

An experiment with two PEs working in parallel reveals that, for the processing of a generation with 100 chromosomes, a time of 223μs was demanded, achieving a time reduction of approximately 54%. This two PEs implementation, using the same device, demands 6,966 logic elements and 36,300 bits of memory, occupying 65% and 3% of available resources, respectively.

Other important issue is the Speedup [12], which is the ratio between the time of execution with one PE and the parallel time of execution with two or more PEs. It is fundamental to analyze the performance of parallel machines. The tests done using this architecture with two PEs achieves around of 85% of Speedup, close to ideal value of 100%.

6 Conclusions and Future Work

The main contribution of this work is the proposal and implementation of a reconfigurable parallel architecture, using Genetic Algorithms and applied to the synthesis (Boolean minimization) of combinational digital circuits.

With the processing in parallel way is to be possible a significant execution time reduction.

In this implementation is observed that approximately 89% of a generation processing time is spent with the chromosome fitness processing. This justifies the PE parallelization approach, where several PEs process the individual fitness in parallel way, regarding to CU other sequential operations processing, such as the selection function operation, the crossover operation and the mutation operation.

In the experiments using 2 PEs, we observe a performance improvement of 54%, or a speedup of 1.85, reaching 92% of the ideal value. However, we will only be able to affirm the evolution of performance profit when the results will be more consistent.

With the addition of one PE in the architecture, we observe a demand of 17% more logic elements. These additional device logic elements are used to implement the additional PE, the interconnection of many components for the

parallel buses, and principally, the memories division in two separated modules. This division avoids a possible bottleneck in input/output data transfer with several PEs architecture.

This architecture explores many levels of parallelism. The first level can be observed in several PEs operating in parallel way. The second level can be observed in geometric set of 5×5 logic cells internally of each PE. Since each gene configures just one individual LC, the complete configuration is made in parallel. By other hand, since 5 logic expressions are simultaneously generated to each input element, a parallel processing is also observed. The last level can be observed in the operation proceeding of the "best chromosome" unit. This operation determines the best chromosome, while the others units still processing its operations simultaneously. Finally, the VHDL programming generates a digital circuit implementation that enables a parallel execution of an algorithm. The similar software solution can not be executed in same way.

Although several improvements can be made in the implemented architecture, the partial obtained results demonstrate the correction of the approach adopted.

Some aspects deserve special attention for the future improvements:

- Increase the number of PEs, to get greater performance;
- Adapt the parallel architecture to be possible working with more complex problems;
- Optimize the circuits to consume less resources of FPGA device.

Finally, this work demonstrates that several positive factors like the processing power of the Parallel Computation, the versatility of Reconfigurable Computation and the robustness of Genetic Algorithm, can be used to created powerful tools to solution of many complex problems. Especially in Engineering and Sciences problems were the processing time is critical restriction.

7 Acknowledgment

This work was partially supported by the Brazilian National Research Council – CNPq, under research grant no. 305720/04-0 to H.S. Lopes.

References

1. Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*, vol. 1, Kluwer Academic, Norwel (2001).
2. Coello-Coello, C.A., Aguirre, A.H.: On the use of a population-based particle swarm optimizer to design combinational logic circuits. In: *Proc. NASA/DoD Conference on Evolvable Hardware* (2004) 183–190.
3. Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., White, A.: *Sourcebook of parallel computing*. Morgan Kaufmann, San Francisco (2003).
4. Goldberg, D.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading (1989).

5. Graham, P., Nelson, B.: A Hardware genetic algorithm for the traveling salesman problem on Splash 2. In: *Field-Programmable Logic and Applications*, Springer-Verlag, Berlin (1995) 352–361.
6. Hartenstein, R.: A Decade of reconfigurable computing: a visionary retrospective. In: *Proc. IEEE Conf. on Design, Automation and Test in Europe* (2001) 642–649.
7. Holland, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, East Lansing (1975).
8. Lysaght, P., Rosenstiel, W.: *New algorithms, Architectures and Applications for Reconfigurable Computing*. Springer, New York (2005).
9. Gokhale, M., Graham, P.S.: *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Springer, Berlin (2005).
10. M. Matsumoto, Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulations* (1998) 3–30.
11. Miller, J.F., Thomson, P.: Cartesian genetic programming. *Proc. 3^{rd} European Conference on Genetic Programming, LNCS* **1802** (2000) 121–132.
12. Murdocca, M.J., Heuring, V.P.: *Principles of Computer Architecture*. Prentice Hall, New Jersey (2001).
13. Pedroni, V.A.: *Circuit Design With VHDL*. MIT Press, Cambridge (2004).
14. Sekanina, L., Ruzicka, R.: *Easily testable image operators: the class of circuits where evolution beats engineers*. In: *Proc. NASA/DoD Conference on Evolvable Hardware*, (2003) 135–144.
15. Yue, K.K., Lilja, D.J.: Designing multiprocessor scheduling algorithms using a distributed genetic algorithm system. *Evolutionary Algorithms in Engineering Applications* **33** (1997) 39–40.
16. Zhang, Y., Smith, S.L., Tyrrell, A.M.: Digital circuit design using intrinsic evolvable hardware. In: *Proc. NASA/DoD Conference on Evolvable Hardware* (2004) 55–62.