Analog Integr Circ Sig Process
DOI 10.1007/s10470-013-0059-1

Parallel algorithm for evolvable-based boolean synthesis on GPUs

Jaime Vitola · Adriana Sanabria · César Pedraza · Johanna Sepúlveda

Received: 30 June 2012 / Revised: 26 November 2012 / Accepted: 23 February 2013

Abstract The use of evolutionary algorithms in the boolean synthesis is an attractive alternative to generate interesting and efficient hardware structures, with a high computational load. This paper presents the implementation of a parallel genetic programming (PGP) for boolean synthesis on a GPU-CPU based platform. Our implementation uses the island model, that allows the parallel and independent evolution of the PGP through the multiple processing units of the GPU and the multiple cores of a new generation desktop processors. We tested multiple mapping alternatives of the PGP on the platform in order to optimize the PGP response time. As a result we show that our approach achieves a speedup up to 41 compared to CPU implementation.

Keywords Evolutionary algorithms · Boolean synthesis · GPU

1 Introduction

One of the main goals in combinatorial synthesis consists of finding compact boolean expressions in the sum of products (SOP) form with the smallest possible number of variables and terms. Boolean algebra techniques such as Karnaugh maps, Quine-McCluskey algorithm and Reed-Muller algorithm offers a way to find compact expressions but in some cases an optimal structure depends on the designer's experience, resulting in non-optimal or inadequate expressions. Also, these algorithms have disadvantages such as exponential complexity, lack of restrictions management, and multiple solutions.

Bio-inspired algorithms are an alternative to create new structures of combinatorial circuits that can not be obtained with the traditional methods and to add some restrictions to the design such as delay, area, etc. These designs have a very low limited number of variables [9] and they are mainly oriented to obtain a few basic structures. Nicholson [15] has used Simple Genetic Algorithms (SGAs) with a fixed length representation for small problems. Also, Kajitani [11] has worked with Variable-length Genetic Algorithms (VGAs) evolving up to 6-bit problems, Aguirre et al. [1] used tree-based genetic programming (GP) for evolving small circuits, Xu [23] has worked with adaptive immune GA obtaining only 4-variable circuits, as well as Coello with ant colony algorithms [5]. Other techniques use cartesian genetic programming (PGP) as a strategy in a multiobjective algorithm [3, 10, 24].

Others authors have proposed a platforms to use reconfiguration techniques with hard-time restrictions due the high reconfiguration latency [14, 20, 21]. Due the high computational requirements for implementing any bio-inspired algorithm, response times usually are too high and only small circuits can be created in reasonable time [12, 22]. Some of these authors have made efforts to reduce the response time by using different platforms such as ultimate CPUs, Field Programmable Gate Arrays (FPGAs) [22], computer clusters [16, 17] and Graphics Processing Units

J. Vitola · A. Sanabria · C. Pedraza (✉)
Universidad Santo Tomás, Cra. 9 No. 51-11, Bogotá, Colombia
e-mail: cesarpedraza@usantotomas.edu.co

J. Vitola
e-mail: jaimevitola@usantotomas.edu.co

A. Sanabria
e-mail: adrianasanabria@usantotomas.edu.co

J. Sepúlveda
University of Sao Paulo Microelectronics Laboratory LME, Sao Paulo, Brazil
e-mail: jsepulveda@lme.usp.br

Published online: 12 March 2013

(GPUs) [13], allowing to implement parallel versions of the bio-inspired algorithms.

A technique called virtual reconfigurable circuit (VRC) on Field Programmable Gate Arrays was proposed by Sekanina [19, 22] and similar to the one developed by Torresen [7, 8], which uses registers as reconfigurable elements to change the functionality of a given circuit. These works showed high performance but caused a high consuming resources, low frequency response due the critical path and high cost of the project.

To solve the problem of the response time of creating digital circuits by using parallel genetic programming (PGP), we have developed a parallel genetic program in a multi-GPU + CPU architecture. To find the best configuration of the algorithm in the parallel platform, the island model were adopted and different schemes for building it on the GPU-CPU based platform were tested.

2 Genetic programming and boolean synthesis

Evolutionary algorithms are computational techniques that Perform a heuristic search based on the principles of the natural evolution, by means of the natural selection. The search space (population) is constituted by a set of solutions called individuals. The initial population is constituted by random generated individuals. In each step, the individuals are evaluated according to the fitness function which determines the adaptation degree. Some individuals of the population are selected and employed to create new individuals through variation operators. The most adapted individuals have higher probability to be part the new population. After some generations is expected to find one or more individuals that satisfy the fitness function.

The way a logic circuit is represented using a bit vector to be used in the evolution process [18] must be ensure to represent all the possible solutions to the problem, and must avoid to the crossover and mutation operators not to generate infeasible individuals. The 2-D tree representation used here is shown in Fig. 1. Each cell has 3 functions f and 4 input variables v coded in binary.

Equation 1 shows the fitness function for the genetic program. Constants ω_1, ω_2 and ω_3 are used for establishing the weights of the parameters that will determine the fitness function. The double-summation term calculates the number of matches of the individual X for all the possible combinations at the output with the objective function Y; the $P(X)$ function is used for calculating the number of logic gates of an individual taking into account some of the *introns* or segments of the genotype string that will not have any associated function and that do not contribute to the result of the logic circuit that they represent. The function $L(X)$ is used for determining the critical path of the circuit. The m constant is the number of outputs in the

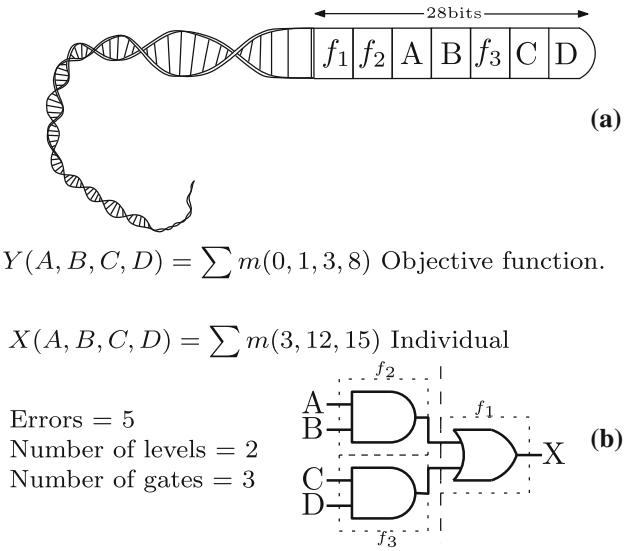

Fig. 1 Cell-based structure representation

circuit and n the number of possible combinations of inputs in the circuit.

$$fitness = \omega_1 . \left[\sum_{j=1}^{m} \sum_{i=1}^{n} Y(j,i) - X(j,i) \right] + \omega_2 . P(x) + \omega_3 . L(x) \quad (1)$$

In order to keep the quality of the individual in the population by means of the diversity, the selection and mutation operators were implemented in the algorithm.

2.1 Parallel genetic programming and the island approach

The evolutionary algorithms are highly parallelizable because the evaluation of the fitness function of each individual of the population at each iteration is independent. The island model is a way where the initial population is splitted into subpopulations, able to evolve in a parallel and an independent way according to a common fitness function. These subpopulations are not completely isolated, once they are connected through a ring, torus or hypercube network topology.

After some iterations it occurs an exchange of individuals of each subpopulation with their neighbours through a new operator called migration. The number of iterations is known as migration intervals. The size of the migrated population and the size of each subpopulation remains an open problem [6].

3 GPU and CPU implementation

3.1 GPU

One of the most interesting applications to be implemented on GPUs are bio-inspired algorithms. The intrinsic parallelism of

these algorithms allows to the designers easily map them on massive parallel platforms such as GPUs. Afterwards are shown the most important aspects in the implementation of the PGP on a GPU-CPU platform. Two main parts can be identified to implement the system on GPUs: (1) The random number generator; and (2) the GP. The GP requires random numbers for generating the initial population, to mutate and cross the individuals. Because the GPU cannot generate random numbers by using C classical libraries, it was important to find another way to get them. The number generation and later communication from the CPU to the GPU is not viable because it will increase the system latency. To solve this problem a Mersenne-twister algorithm is executed on the GPU before the the *kernel−GP* in order to make a buffer of random numbers on its own global memory.

3.2 Kernel structure

Figure 2 shows our implementation of the GP on the graphics device. A thread *t* executes a *kernel−GP* (a common segment of code that process different data in each instance), and generates a μ-population, it performs the operators of selection, mutation and crossover operators. After *P* generations, *M* individuals will be transferred to the global memory and then to the host device (CPU system). *P* is known as the frequency of migration and the *M* number is called the migration factor.

It can be observed that each thread can cooperate with other threads inside the same block through the shared memory, sharing the best individuals and improving the efficiency of the GP.

3.3 CPU

One technique to improve the performance of a GP is implement it in parallel form. Nowadays devices such as

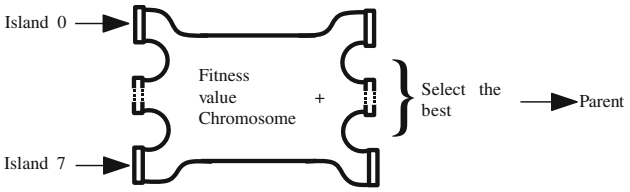

Fig. 3 Processes scheme for evolving individuals in different processor desktop cores

CPUs offers multiple processing cores that can be exploited for this purposes. By launching multiple processes on the CPU, identical copies of the evolutionary algorithm evolved individuals such as the island model. The best *m* individuals were selected and migrated each *p* iterations via pipes that connect the parent process with the rest of the processes. Figure 3 shows the communications scheme used to run the algorithm in the CPU platform.

However the Amdahl law limits the performance improvement of the algorithm [2], because when an element has been improved, the total performance is upgraded in a fraction of the time spent by the component.

In order of mitigating the law's effect, the Intel Core i7 processors employ the Turbo boost technology which allows to increase the main clock frequency to the cores during small periods of time. These overclock is made according to the the quantity of active cores, the estimated current consumption, the estimated power consumption and the processor temperature. With this function in the experiments described here it was obtained an improvement between 3 and 12 %, however the measurements are concentrated between 4 and 6 %.

4 Performance evaluation

The PGP were tested in a Multi-GPU architecture based on two NVIDIA Tesla devices, each one made by 448 CUDA

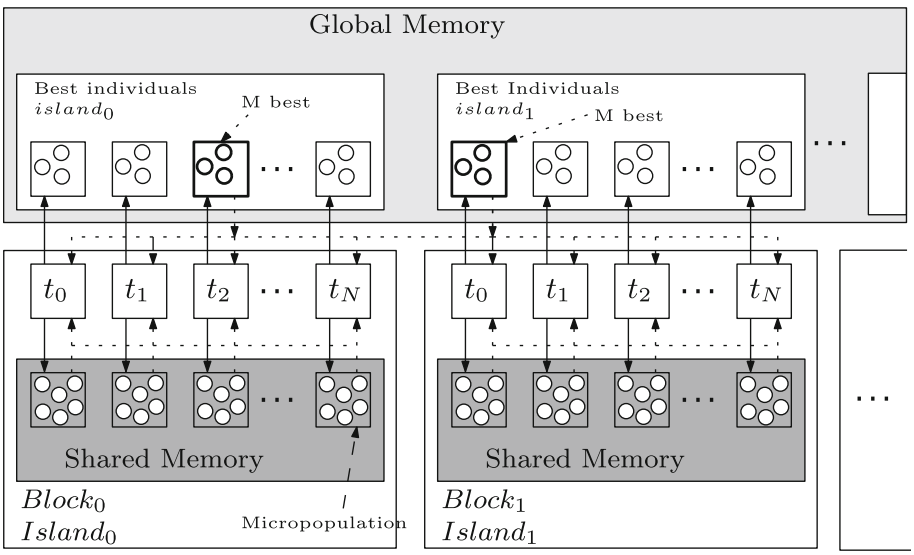

Fig. 2 GPU implementation of the evolvable algorithm

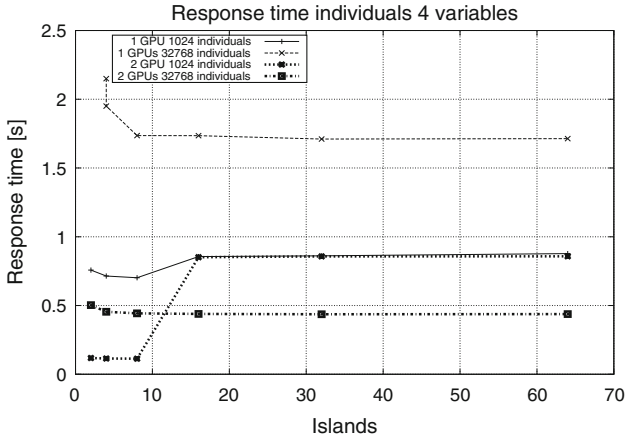

Fig. 4 Response time with 1 and 2 GPUs, 4 variables and different number of individuals

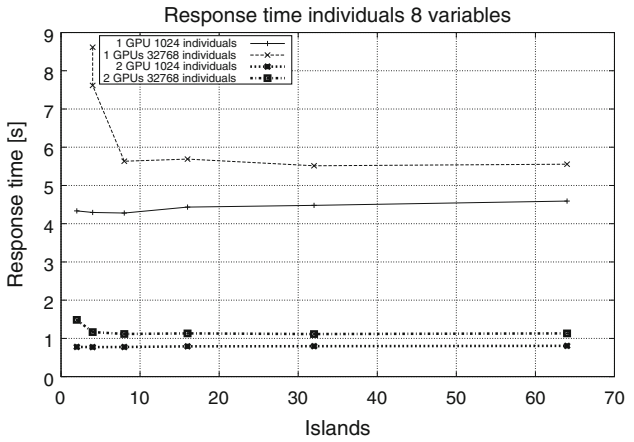

Fig. 5 Response time with 1 and 2 GPUs, 8 variables and different number of individuals

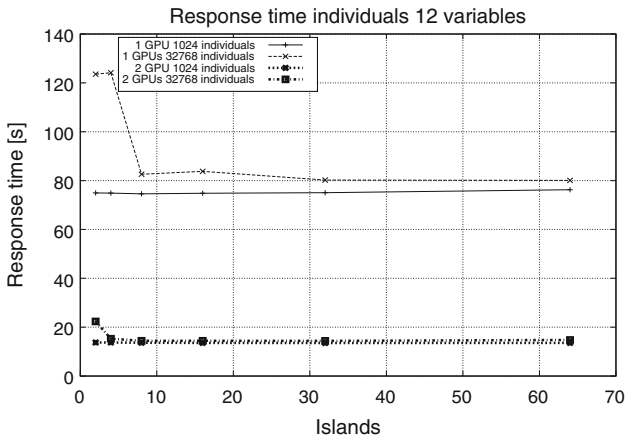

Fig. 6 Response time with 1 and 2 GPUs, 12 variables and different number of individuals

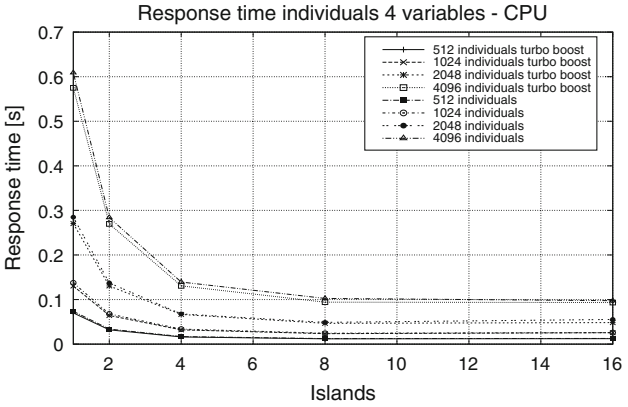

Fig. 7 Response time CPUs, 4 variables and different number of individuals

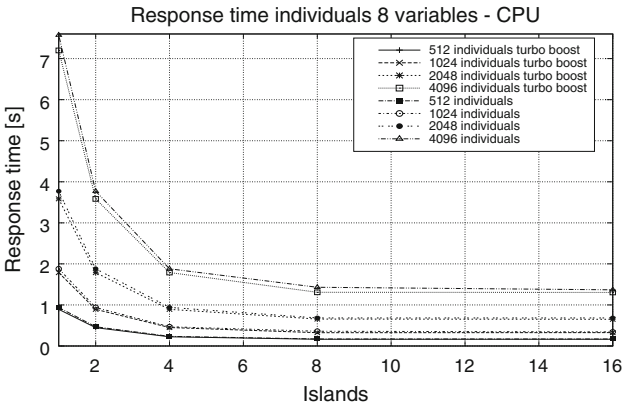

Fig. 8 Response time CPUs, 8 variables and different number of individuals

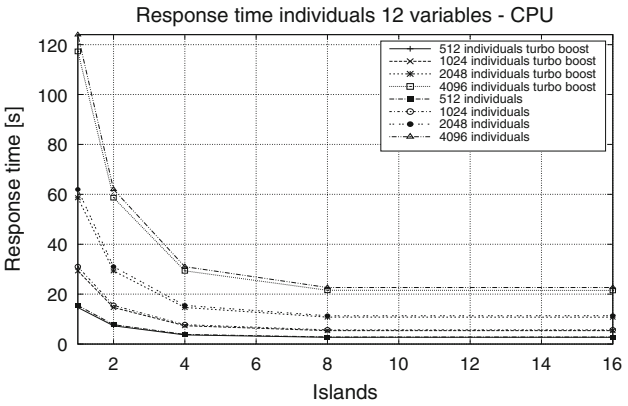

Fig. 9 Response time CPUs, 12 variables and different number of individuals

cores, working at 1.15 GHz and a DDR5-3GB global memory. On the other side, a Intel Core i7 CPU 4GB RAM platform were used to test the performance of the PGP on this kind of architectures.

Table 1 Speedup and response time for GPU vs GPU shared configuration

Blocks	GPU shared 1,024 indiv vs GPU	GPU shared vs GPU 32,768 indiv	RT/indiv 1 GPU (μs)	RT/indiv GPU shared (μs)
2	1.1001	1.8010	1,230	682
4	1.1100	1.8105	843	466
8	1.1200	1.8200	799	440
16	1.1540	1.8200	801	440
32	1.3101	1.8205	800	440
64	1.3200	0.8315	825	451

Table 2 Speedup and response time for CPU vs GPU configurations

Processes	CPU vs GPU shared 1K indiv	CPU vs GPU 32K indiv shared	CPU vs GPU 1K indiv	CPU vs GPU 32K indiv	RT/ Indiv CPU (μs)
1	2.1353	41.9185	1.9413	23.2751	28,600
2	1.0685	30.7348	0.9627	16.9759	1,430
4	0.5383	16.2565	0.4806	8.9553	7,160
8	0.3978	12.0240	0.3447	6.6066	5,300
16	0.3998	11.9683	0.3052	6.5742	5,260

4.1 Experiment setup

Several scenarios have been tested with different input parameter configurations: (1) number of input variables (4, 8 or 12, corresponding to a comparator problem of 2, 4 and 6 bits), (2) population size (512, 1,024 or 2,048) and (3) number of threads and islands running the experiment, ranged from 1 to 16 in the CPU, and 1 to 64 in the Multi-GPU platform. The first and second parameters determine the size of the problem. The last one gives an idea about the scalability of the system. Results of these configurations were compared to other works when dedicated hardware with Field Programmable Gate Array devices were developed.

4.2 Response time

Figures 4, 5 and 6 show the response time for the Multi-GPU platform using 1 and 2 GPUs with different numbers of islands and variables with 1,024 and 32,768 individuals during 100 generations.

These results show the high performance of the platform for the algorithm. This experiment demonstrates that the response time depends on the size of the problem because individuals complexity is increased exponentially with the number of variables of the problem, and they had to be evaluated by software. In contrast, response time in [17]

Table 3 Speedup for CPU with and without turbo boost - 12 variables

Islands	Indiv	Time TB	Speedup TB	Time	Speedup	Improvement (%)
1	512	14.6482	1	15.4918	1	5.4452
2	512	7.3266	1.9993	7.7456	2.0006	5.4097
4	512	3.6667	3.9948	3.8756	4.0007	5.3892
8	512	2.6934	5.4383	2.8347	5.47126	4.9835
16	512	2.6961	5.4329	2.8343	5.4730	4.8733
1	1,024	29.3095	1	31.0001	1	5.4536
2	1,024	14.6615	1.9990	15.4963	1.9996	5.3867
4	1,024	7.3294	3.9988	7.7497	4.0006	5.42297
8	1,024	5.3867	5.4410	5.67026	5.47037	4.9993
16	1,024	5.3913	5.4364	5.6693	5.4603	4.9028
1	2,048	58.6177	1	61.9610	1	5.3958
2	2,048	29.3135	1.9996	31.0192	1.9966	5.4988
4	2,048	14.6599	3.9985	15.5009	3.9946	5.4260
8	2,048	10.7777	5.4387	11.3294	5.4674	4.8693
16	2,048	10.76422	5.4456	11.3521	5.4661	5.1787
1	4,096	117.2707	1	123.9318	1	5.3748
2	4,096	58.6455	1.9996	61.9928	2.0008	5.3994
4	4,096	29.3497	3.9956	31.0108	3.9957	5.3562
8	4,096	21.5426	5.4436	22.6566	5.48167	4.9169
16	4,096	21.4975	5.45507	22.6500	5.47263	5.0882

has not a high dependency of the size of the problem. Also, the figures show the response time when a high number of individuals are evolved (32,768), it is demonstrated that using two GPUs is more suitable for this purposes. Despite population were incremented from 1,024 to 32,768 individuals, response time did not increase dramatically, this is because communications time is included in both scenarios.

In the same way, Figs. 7, 8 and 9 show the same scenario when the algorithm is executed in the CPU platform, launching 1 up to 16 processes on each one of its cores. Again, it is demonstrated the high dependency of the number of variables for the algorithm. Also, it can be notice a small improvement of the algorithm when the Intel Turbo Boost were activated when a problem of any number of variables were evolved. In the same way, it is shown the response time becomes constant when 8 or more processes were launch, because the 4 HT cores of the Intel Core i7.

4.3 Speedup

Tables 1 and 2 show the speedup numbers for different configurations of the algorithm on GPU and CPU platforms. In Table 1 can be observed the speedup number when compared the performance in 1 GPU vs 2 GPUs, it is clear a better performance on multiple GPUs when a high number of individuals are present in the population. Also, it is shown the time response for evolving one individual, which is around some hundreds microseconds.

In the other case, the algorithm were tested in the CPU architecture with different number of processes. In this case it can be observed a high performance of GPUs vs CPUs when many individuals are evolved. Also, the response time for individual was calculated, resulting in some hundred of microseconds. It can be deduced a higher response time in CPU configuration when large populations of individuals are evolved.

The Table 3 shows the results obtained for evolving problems of 12 variables and a population size of 512–4,096 individuals. The values of execution time and speedup with Turbo boost (*Time TB* and *speedup TB*) and without Turbo Boost (*Time* and *speedup*) respectively. The last column shows the improvement in the execution time with and without the Turbo boost funcionality. For the others cases evaluated the results were similar.

As shown in Table 3, the results of using the Intel Turbo Boost Technology are similar to the obtained in [4]. An improvement up to 5.1% shows the advantages of using this technology.

The Figs. 10, 11 and 12 show the speedup for the PGP executed on CPUs and 4, 8 and 12 variables.

Finally, Figs. 10, 11 and 12 show the speedup number for the CPU scenarios. This metric were calculated comparing the performance when one process were launch against 2, 4, 8 and 16 launched processes. In all cases the

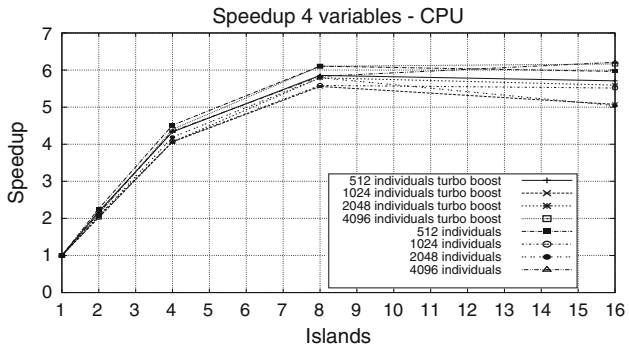

Fig. 10 Speedup CPUs, 4 variables and different number of individuals

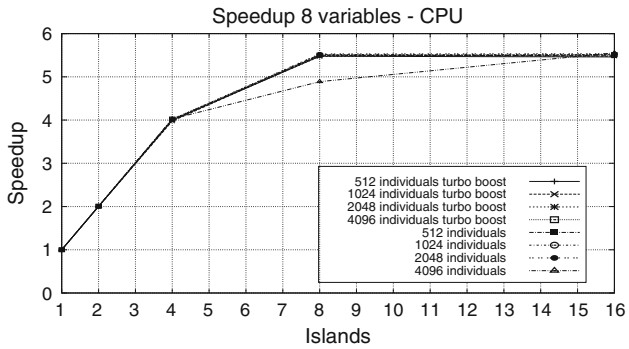

Fig. 11 Speedup CPUs, 8 variables and different number of individuals

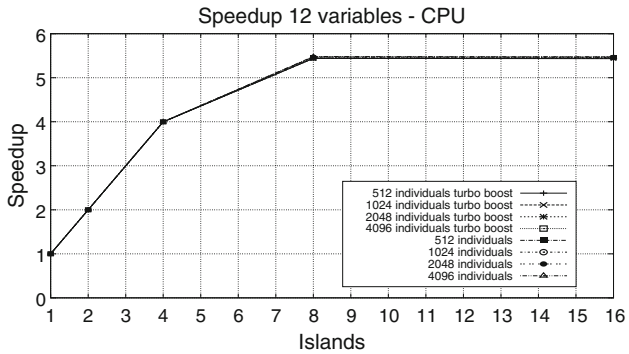

Fig. 12 Speedup CPUs, 12 variables and different number of individuals

speedup is about 5, which demonstrates the use of the 4 cores with the Hyper Threading technology.

5 Conclusions and future work

This paper presented a novel and fast way to evaluate individuals in an evolvable algorithm on a Multi-GPU and CPU platforms and results were compared to a FPGA-based accelerated work. Despite there are other works related with genetic programs over GPUs [?], this paper showed an specific application of them to the acceleration process in creation of digital hardware.

Results showed a speedup up to 41 evolving upto 32 K individuals on the Multi-GPU system compared to CPU, even when 4 cores were used in the last one. When the number of individuals is close to 1,024, the GPU-shared architecture showed a low performance compared to the CPU because the communications processes become important. When compared single GPU, Multi-GPU configuration showed a speedup up to 1.3 with large populations. In contrast, small populations are faster evolved in one GPU. In the same way, CPU improvements up to 5 % with Turbo Boost Technology were achieved.

Significant speedup numbers were obtained in [17] due individuals had been directly tested in hardware, getting a combination of their true table on each cycle of the system clock. Tests proved that the algorithm is more effective for 4-bit and 8-bit problems. 12-bit problems in GPU had excellent performance, but because the search space is too long, converging to a suitable solution was difficult for the algorithm. This problem could be solved as future work with some improvements in terms of GPUs memory optimization.

Acknowledgments This study was supported by the Santo Tomàs University FODE-IN program, project number 6011021101.

References

1. Aguirre, A., Coello, C., & Buckles, B. (1999). A genetic programming approach to logic function synthesis by means of multiplexers. *Proceedings of the First NASA/DoD Workshop on Evolvable* pp. 46 – 53.
2. Amdahl, G.M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In: *Proceedings of the April 18-20, 1967, spring joint computer conference, AFIPS '67 Spring*, (pp. 483–485). ACM, New York (1967). doi: 10.1145/1465482.1465560. URL http://doi.acm.org/10.1145/1465482.1465560.
3. Bremner, P., Samie, M., & Pipe, A. (2011). *Multi-objective optimisation of cell-array circuit evolution*. (CEC), 2011 IEEE (pp. 440–446).
4. Charles, J., Jassi, P., Ananth, N.S., Sadat, A., & Fedorova, A. (2009). Evaluation of the Intel® Core™ i7 Turbo Boost feature. In: 2009 IEEE International Symposium on Workload Characterization (IISWC), (pp. 188–197). IEEE. doi:10.1109/IISWC.2009.5306782.
5. Coello, C., Zavala, R., & García, B. (2000). Ant colony system for the design of combinational logic circuits. *Evolvable Systems: From Biology to Hardware* (pp. 21–30).
6. Eiben, A.E., & Smith, J. (2010). Introduction to evolutionary computing (Natural Computing Series). Heidelberg: Springer.
7. Glette, K., & Torresen, J. (2005). A flexible on-chip evolution system Implemented on a Xilinx Virtex-II Pro Device. *Evolvable Systems: From Biology to Hardware* (pp. 66–75).
8. Glette, K., Torresen, J., & Yasunaga, M. (2009). Online evolvable pattern recognition hardware. In: *Evolutionary Image Analysis and Signal Processing*, (pp. 41–54) Heidelberg: Springer.
9. Goldberg, D., & Holland, J. (1988). Genetic algorithms and machine learning. *Machine Learning, 3*(2): 95–99.
10. Harding, S., Miller, J.F., & Banzhaf, W. (2010). Developments in Cartesian genetic programming: self-pmodifying CGP. *Genetic Programming and Evolvable Machines, 11*(3-4):397–439 doi: 10.1007/s10710-010-9114-1.
11. Kajitani, I., Hoshino, T., Iwata, M., & Higuchi, T. (1996). Variable length chromosome GA for evolvable hardware. In: *Proc. of the 3rd Int. Conf. on Evolutionary Computation*, (pp. 443–447). Japan: Nagoya.
12. Koza, J., Keane, M., Streeter, M., Mydlowec, W., & Yu, J. (2005). Genetic programming IV: routine human-competitive machine. Burlington: Morgan Kaufmann.
13. Luong, T.V., Melab, N., & Talbi, E.G. (2010). GPU-based island model for evolutionary algorithms. In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, (p. 1089). New York: ACM doi:10.1145/1830483.1830685
14. Moreno, J., Thoma, Y., & Sanchez, E. (2006). POETIC: a hardware prototyping platform with bio-inspired capabilities. In: *Mixed Design of Integrated Circuits and System, 2006*. MIXDES (pp. 363–368).
15. Nicholson, A. (2000). Evolution and Learning for Digital Circuit Design. In: *Proceedings of Genetic and Evolutionary Computation Conf*, (pp. 519–524).
16. Pedraza, C., Castillo, E., Castillo, J., Camarero, C., Bosque, J., Martinez, J., & Menendez, R. (2008). Cluster architecture based on low cost reconfigurable hardware. In: *Field Programmable Logic and Applications*, FPL 2008. International Conference on, (pp. 595–598) Heidelberg.
17. Pedraza, C., Castillo, J., Martínez, J., & Huerta, P. (2011) Genetic Algorithm for Boolean minimization in an FPGA cluster. *Journal of Supercomputing* 58(2):244–252.
18. Rothlauf, F. (2006). *Representations for genetic and evolutionay algorithms*. Heidelberg: Springer.
19. Sekanina, L. (2009). Evolvable Hardware: From Applications to Implications for the theory of computation. In: *Unconventional Computation,Lecture Notes in Computer Science, vol. 5715/2009*, (pp. 24–36) Heidelberg: Springer.
20. Thoma, Y., Sanchez, E., & Hetherington, C. (2004). Prototyping with a bio-inspired reconfigurable chip. In: *15th IEEE International Workshop on Rapid System Prototyping*.
21. Upegui, A., & Sanchez, E. (2005). Evolving hardware by dynamically reconfiguring Xilinx FPGAs. *Evolvable Systems: From Biology to Hardware, Lectures notes in computer science, 3637/2005*, 56–65.
22. Vasícek, Z., & Sekanina, L. (2008). Hardware accelerators for cartesian genetic programming. *Genetic Programming, Lecture Notes in Computer Science, 4971/2008*, 230–241.
23. Xu, H., Ding, Y., & Hu, Z. (2009). Adaptive immune genetic algorithm for logic circuit design. In: *Proceedings of the first*

ACM/SIGEVO Summit on Genetic and Evolutionary Computation, (pp. 639–644). ACM.
24. Zhao, S., & Jiao, L. (2006). Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm. *Genetic Programming and Evolvable Machines, 7*(3), 195–210 doi:10.1007/s10710-006-9005-7.

Jaime Vitola received the B.S. degree in electronics engineering from the Santo Tomas University in Bogota, Colombia. The M.S degree in Electronics Engineering from the Distrital Francisco Jose de Caldas University in Bogotá.His interest are digital signal processing, advanced digital design with FPGAs.

Adriana Sanabria has a B. S degree in electronics engineering from the Santo Tomas University in Bogota, Colombia. Currently she is an student at the M.S. program in the Instituto Nacional de Astrofísica, Óptica y Electrónica INAOE in Puebla Mexico.His interest are microelectronics, digital signal processing and advanced digital design with FPGAs.

César Pedraza received the B.S. degree in electronics engineering from the Universidad Santo Tomas, Bogota Colombia. Also, he has a M.S. degree from the Universidad de Los Andes, Bogota Colombia and a Ph.D. degree in Informatics Engineering from the Rey Juan Carlos University in Madrid, Spain. His current research are bio-inspired algorithms, digital signal processing and parallel computing.

Johanna Sepúlveda received the B.S. degree (summa cum laude) in electronics engineering from the Colombia National University, Bogota, Colombia, in 2004. The M.S. degree in electrical engineering in 2006 and the Ph.D. degree in electrical engineering from the Sao Paulo University, Sao Paulo, Brazil, in 2011. Currently, she holds a Postdoctoral position in the microelectronics laboratory at the Sao Paulo University, Sao Paulo, Brazil.Her current research interests include the design of high performance SoC communication structures and the inclusion of Quality-of-Service and security in embedded systems.