FPGA Implementation of a Cellular Compact Genetic Algorithm

Yutana Jewajinda
National Electronics and Computer Technology Center
National Science and Technology Development Agency
Bangkok, Thailand
yutana.jewajinda@nectec.or.th

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
prabhas@chula.ac.th

Abstract

This paper presents a cellular compact genetic algorithm (CCGA) for evolvable and adaptive hardware. The CCGA has cellular-like structure which is suitable for hardware implementation. The CCGA is developed from compact genetic algorithm (CGA) and parallel estimation of distribution algorithm (EDA). The concept and algorithm of the CCGA are presented. The standard test functions are selected to measure the effectiveness of the CCGA. The experimental results significantly shows that the CCGA outperforms the normal compact GA and deliver compatible results to the cooperative compact genetic algorithm while employs only one type of cell. The implemented hardware in FPGA demonstrates the feasibility to use this new kind of genetic algorithm to evolvable and adaptive hardware.

1. Introduction

Evolvable hardware (EH) is a research area in the field of evolutionary computation (EC). EH is the integration of evolutionary computation and programmable hardware devices. The objective of evolvable and adaptive hardware is to create "autonomous" reconfiguration of hardware structures in order to improve performance [1]. With the use of evolutionary computation and reconfigurable device like FPGA, evolvable hardware has the capability to autonomously change its hardware architecture and function. Recent research trend directs toward functional approaches for the design of extrinsic and intrinsic EH [2-6].

The key concept of our focused evolvable and adaptive hardware is to regard the configuration bits of programmable hardware architecture as the chromosomes of Genetic algorithm (GA) [1]. By optimizing a fitness function to achieve a desired hardware function, the GA becomes a means of autonomous hardware configuration. There are a number of methods and techniques that propose to apply the Genetic Algorithm (GA) and Evolutionary Algorithms (EA) to be implemented in hardware for evolvable hardware (EH) and adaptive hardware, especially to be implemented into FPGAs [7,9-10]. However, in order to accomplish the intrinsically on-line evolving in hardware pose a challenging question of how to modify or invent efficient and improved GA or EA algorithms for hardware implementation [9].

The compact genetic algorithm (CGA) is a kind of the probabilistic model-building genetic algorithms (PMBGAs) or the estimation of distribution algorithms (EDA) [11]. The compact GA operates on probability models or probability vectors by replacing the crossover and mutation operators with the probability model estimation. The CGA) can be efficiently implemented in digital hardware [9, 10]. Even though CGA has advantage for hardware implementation; however, the CGA lacks sufficient search power for real world EH applications that requires more accuracy and faster processing time. Therefore, the CGA has been improved by adding more techniques like elitism, mutation, and champion resampling. This modified CGA is called *CGA or *CGA family [9].

The parallelization of GA has been the active research topic using high performance computer systems [12]. The parallelized GA can be efficiently implemented in hardware with more available hardware resources in current FPGA devices. Parallel GA has been modified and implemented in FPGA to offer more search power in hardware [10].

In this paper, we present the cellular compact genetic algorithm (CCGA) and explore its hardware implementation. CCGA is based on parallel genetic algorithm [12]. It is similar to cooperative compact genetic algorithms [10]. However, cellular CGA is more suitable for hardware implementation since it has a cellular-like structure which is a two dimensional array with a uniform cell type. In addition, CCGA is characterized as a parallel EDA [13] with improvement on probability model recombination.

978-0-7695-3166-3/08 $25.00 © 2008 IEEE
DOI 10.1109/AHS.2008.34

The rest of this paper is organized as follows. Section 2 describes the cellular compact genetic algorithm. In Section 3, the hardware design of CCGA is presented. Section 4 presents FPGA implementation results. The paper concludes with a summary in Section 5.

```
1. Initialize probability vector
     for i := 1 to L do p[i] := 0.5;
2. Generate two individuals from the vector
     a := generate(p);
     b := generate(p);
3  Let them compete
     Winner, loser := evaluate(a, b);
4. Update the probability vector toward the better one
     for i := 1 to L do
     if winner[i]  != loser[i] then
        if winner[i] = 1 then p[i] += 1/N
        else p[i] -= 1/N
5.  Check if the probability vector has converged
    for i := 1 to L do
     if p[i] > 0 and p[i] < 1 then goto step 2
6.   P represents the final solution
```

Figure 1. Pseudocode of compact GA

2. Cellular Compact Genetic Algorithm

Cellular compact genetic algorithm is developed from compact genetic algorithm [11] and parallel GAs [12]. The concept of cellular compact genetic algorithm is to parallelize or divide a large problem into smaller tasks and to solve the task simultaneously using multiple genetic algorithms. CCGA is different from a traditional parallel GA since it operates on probability vectors. CCGA is a parallel univariate estimation of distribution algorithms (EDAs) that migrates the probability model instead of individuals [13]. CCGA improves model combination through local search by selecting the better model from neighbors to be combined with the inner model of the cell [14]. The CCGA consists of uniform cellular compact genetic algorithm cells connected in a cellular automata space by each CGA cell only exchange probability vectors to its neighbors. In this section, we describe the compact genetic algorithm which is the foundation of CCGA. The topology and the algorithm of CCGA are also described.

2.1. Compact Genetic Algorithm

The fundamental of the CCGA is the compact GA [11]. The compact GA represents the population as a probability distribution over the set of solutions. Thus, the CGA maintains a probability vector which is constantly updated while the CGA operates. At each generation, the two individuals are randomly generated from the probability vector. Then, tournament selection is performed over the two individuals. Each bit of the probability vector is adjusted according to the result of the tournament selection. Eventually, the CGA keeps running until the probability vector is converged. The pseudocode of the compact GA is shown in Fig. 1. The hardware implementation of compact GA consists of each bit represented by a probability vector which is connected to form a chromosome. The basic hardware design of compact GA and its variant can be found in [9].

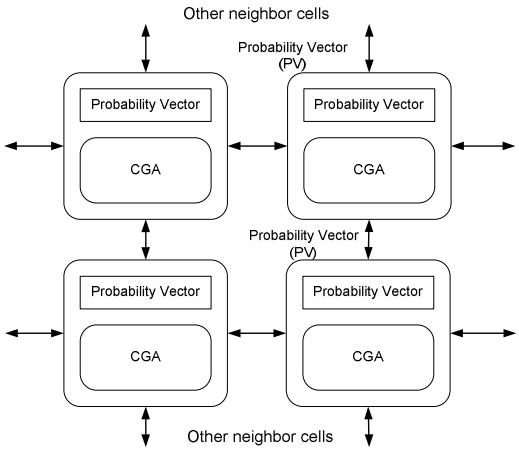

Figure 2. Topology of cellular compact GA

2.2. CCGA Topology

Fig. 2 illustrates the topology of the cellular compact GA. The topology of the proposed CCGA resembles the cellular automata (CA) system that cells only interact with their neighbors [16]. When each local CA cells in cellular automata space operates together, the global states of computation can emerged [16]. With this proposed CA topology, the hardware realization of the algorithm is straight forward and can be practically and efficiently implemented into FPGA because of the architecture of array of logic block [17].

Each coarse grained CCGA cell has a probability vector which represents a sub-population. Every CCGA cell is identical. In Fig. 2, Each CCGA cell with four neighbors exchanges probability vectors and key information between its neighbors. Every CCGA cell keeps adjusting its own probability vector to the better probability. The confidence counter (CC) is introduced to help each cell evaluates how to recombine the probability vectors coming from its neighbors. The key parameters for CCGA topology is the number of the neighbors of each cell.

```
L is chromosome length
N is population size
cc is Confident Counter
CA is Cellular Automata space

for each cell I in CA do in parallel
    Initialize each p[I]
        For  i  := 1  to L do
          [i] = 0.5;
    Initialize cc
          cc := 0;
end parallel for
for each cell  i  in  CA do in parallel
  while not done do
  1.  Generate two individual from the vector
          a := generate ();
          b := generate ();
  2.  Let them compete
          Winner, loser := compete (a, b);
  3.  Update the probability vector toward
      better one and Increment Confidence Counter

          for i := 1 to L do
          if winner[i]  != loser[i] then
            if winner[i] = 1
               cc: = cc + 1;
            then p[i] += 1/N
            else p[i] -= 1/N

  4.  Check if cc reaches a target level then
      Send p and cc to the neighbor cell
  5.  Receives  p and cc from neighbors
  6.  Use the adaptive convex recombination
      with the received p from the neighbor
      and its own p
      6.1  Select the highest cc from neighbors
          cc_max := 0;
          for i := 1 to M do
             if (cc[i] > cc_max )
          cc_max := cc[i];
          p_ccmax := p[i]

      6.2 Convert cc_max to β : 0 ≤ β ≤ 1
          β := MAP( 1/ cc_max)

      6.3. Update p_l with β
          for i := 1 to L do
            p_l[i] := β p_l[i]  + (1 - β ) p_ccmax[i]

  7.  Check if the vector has converged
          for i := 1 to L do
             if p[i] > 0 and p[i] < 1 then
               goto step 1
  8.  p represents the final solution
  end while
end parallel for
```

Figure 3. Pseudocode of cellular compact GA

2.3. CCGA Algorithm

Fig. 3 shows the pseudocode of the cellular compact GA. Each cell of the CCGA has the identical algorithm as shown in Fig. 3. For each cell, one bit of the GA is represented by a probability vector. There are eight steps in the algorithm. The fourth, fifth, and sixth step are added to the standard compact genetic algorithm. At the third step, the confident counter is used to tract the frequency of updating of the probability vector.

At first, the probability vector of each CCGA cell in the cellular automata space is initialized to the mid-point range. In the first and second step, two individuals are generated from the probability vector, then compete similar to a normal compact GA. The probability vector of each cell is updated as shown at step 3 in Fig. 3. When the probability vector is updated, the confident counter is incremented. If the confident counter of each cell reaches a certain level, then the probability vector and the confident counter of each cell are passed to its neighbors.

In fifth and sixth step, once a cell receives the probability vector and confident counter from its neighbor, the cell performs local search by selecting the best probability vector from the incoming vectors. Then, the new inner probability vector is calculated from the adaptive combination weighted by the β value, which derives from the best confident counter shown in step 6.2.

Using β for vector recombination in step 6.3, the CCGA can avoid local minima of the greedy search by shifting search direction gradually toward the better one. This feature of the CCGA contributes to the better performance when compared to the cooperative compact GAs.

Finally, the CCGA keeps running until the probability vector is converged.

The proposed CCGA algorithm is different from the normal compact GA and the cooperative compact GA [14] in four ways:

(1) With uniform cells, the probability vectors are passed directly to neighbor cells.

(2) The confidence toward the better probability vector is calculated as confident counters and passed to neighbor cells. In figure 3, the step 3.2, 4 and 5 are inserted into the normal Compact GA.

(3) Improved probability vector combination is implemented by local search and adaptive combination in step 6 in Fig 3. This combination scheme proposed to provide a solution to the greedy search characteristic of the cooperative compact genetic algorithm [10,14]. The local search is implemented through selecting the best probability vectors among its neighbor and the confident counter that keeps frequency of the updating to the probability vector of each cellular compact GA cell. The higher *confident counter* values contribute to higher

chance to reach the better solution. The probability vector combination refers to the following equation:

$$\mathcal{P}_i^{t+1}(x) = \beta\, \mathcal{P}_i^t(x) + (1 - \beta)\mathcal{P}_r^t(x)$$

Where β is the adaptive weight calculated from the best *confident counter* among neighbors. The better confident counter will provide the lower β which increases the influence of the incoming model from the neighbors.

$\mathcal{P}_i^{t+1}(x)$ is a new inner probability vector of a CCGA cell

$\mathcal{P}_r^t(x)$ is the best incoming probability vector from neighbors

(4) Asynchronous migration rate of probability vector for each CCGA cell using *confident counter*. Since the updating rate of each CCGA cell to its *confident counter* different. This contributes to the different rate to exchange the probability vector.

3. Hardware Design

A CGA cell is designed by adding additional modules to the hardware of the normal compact GA hardware [9]. Fig 4 shows the hardware design of the N-bit module of a CCGA cell. CCGA bit-module is based on the design proposed in [10] integrated with the communication unit (COMM), the confident counter unit (CC) and the probability vector combination unit (VCOMBIN).

In Fig 4, the hardware design consists of four main blocks. The first block is the CCGA bit-module which can be cascaded to form N-bit chromosome. The second block is the CC&COMM unit that has the confident counter (CC) and the communication unit COMM. The third block is the probability vector combination unit VCOMBIN. The fourth block is a simple finite state machine acts as the main controller for the whole block. The detail of these three additional modules is described as follows.

COMM is a finite state machine that controls sending and receiving the probability vector as an 8-bit package between each CCGA cell. For a chromosome of N-bit length, the CCGA needs to have N number of probability vector which each probability vector sizes 8-bit. Thus, for N-bit length chromosome, N packages of 8-bit will be sent and received between each CCGA cell by the COMM unit.

CC is the *confident counter* designed as a 5-bit counter. During fitness evaluation, the counter is incremented every time when the fitness of the winner is better than the current best fitness. The value of the counter is passed to the neighbor CCGA cells with the current probability vector.

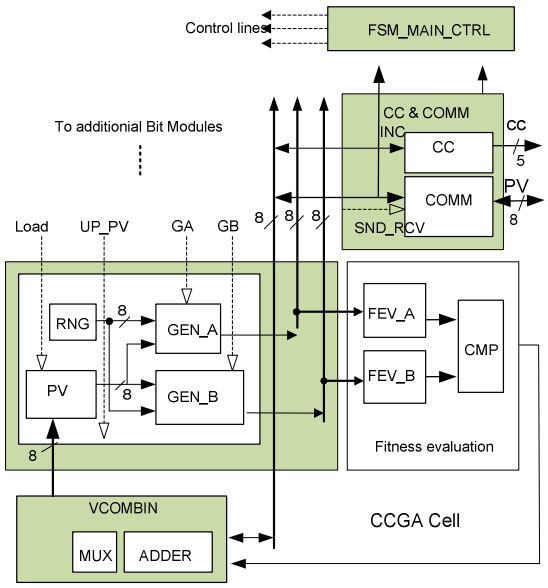

Figure 4. Hardware design of a cellular compact GA cell

VCOMBIN is the hardware block that implements the step 6 of the pseudocode the Fig. 3. A hardware part of the block consists of comparators and multiplexers for comparing incoming *confident counter*. The best confident counter will be selected among the incoming confident counters of the neighbors. The confident counter (cc) is converted to β by using fractional number (1/cc). The multiplication of β with the probability vector is implemented using shift register instead of using multipliers which occupy more hardware resource. With shift register implementation, the β value which is equal to 1/*cc,* will be scaled down to multiple of 2. From equation of vector combination of CCGA algorithm, after multiplication, the value of both probability vectors will be added using 8-bit adder.

FSM_CONTROL is a simple finite state machine that controls the three datapath blocks. The CCGA-bit module takes four clock cycles for generating the two values A and B for tournament selection and updating probability vectors. The COMM module takes sixteen clock cycles for sending and receiving probability vectors; however, the number of clock cycles depends on the size of the chromosome for a specific problem. VCOMBIN takes two clock cycles for latching probability vector to the internal registers and perform shifting and addition.

4. FPGA Implementation results

We implemented the CCGA with two neighbors in Virtex-5 LX50 device. The code was design and coded in

synthesizable Verilog HDL. ModelSim Version 6.2 was used for simulation. Xilinx ISE 9.1 was used for FPGA implementation. For our initial tests of the implementation, "one max" problem with 32-bit was used to verify the operation of the CCGA. The simulation result of "one-max" is shown in Fig. 5. The hardware was also tested with F1 and F2 functions as follows:

$$F1(x) = \sum_{i=1}^{3} x_i \quad when \; -5.12 < x < 5.12$$

$$F2(x) = 100\,(x_1^2 - (x_1^2 - x_2)^2 + (1 - x_1)^2$$

when -2.048 $< x_i <$ 2.048.

In Fig. 5, Fig. 6 and Fig. 7, the simulation results show the comparison between normal compact GA, the cooperative GA and cellular compact GA for OneMax, F1, and F2 functions. The CCGA and CoCGA contain two nodes; each has 32-bit probability vectors while CGA has only one node with one 32-bit probability vector. The performance of CCGA outperforms the normal CGA in term of speed and quality of the search results. CCGA provides at least two times speed up over normal compact GA in Onemax, F1 and F2 test cases as shown in Table 1. In addition, the CCGA provides the compatible speed up to the cooperative compact GA.

Table 2 shows the FPGA implementation of one node and when CCGA is scaled up to four nodes. From Table 2, the speed of the CCGA is not related to the number of the nodes which demonstrates that CCGA can be scaled up to a problem size in FPGA hardware. The comparison of FPGA resources is shown in Table 2. CCGA occupies the same amount of FPGA resources as others CGA. However, it's more practical to FPGA implementation since it has uniform cell type.

The comparison in term of speed and hardware resources to others compact GA implementation is shown in Table 3. CCGA delivers the same speed and requires the compatible hardware resources.

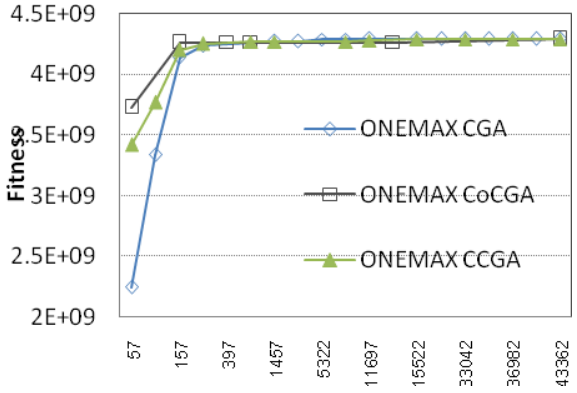

Figure 5. 32-bit "OneMax" simulation results

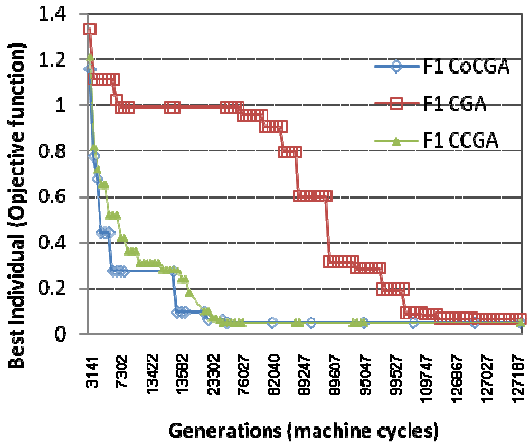

Figure 6. F1 simulation results

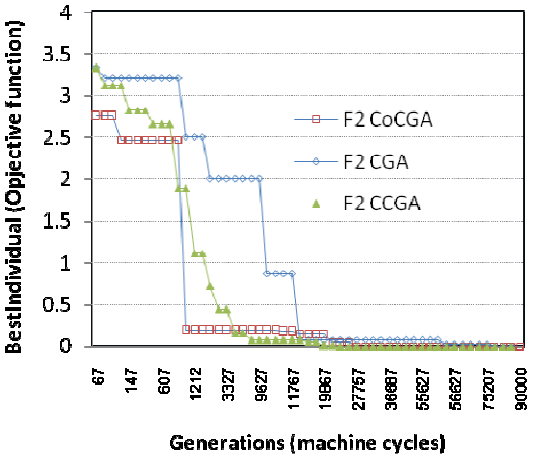

Figure 7. F2 simulation results

TABLE 1 Comparison of the speed up

	OneMax	F1	F2
CGA	43362	126967	80027
CoCGA	11492	25542	27757
CGA	12321	28853	26591
Speedup CoCGA/CGA	3.77	4.97	2.88
Speedup CCGA/CGA	3.51	4.44	3.00

TABLE 2
FPGA HARDWARE RESOURCE XILINX VIRTEX-5 LX50

Network size	FPGA resources for CCGA with 32-bit chromosome on Xilinx Vertex-5 LX50	
		CCGA
1	Slice Registers used Flip-Flops	621
	Slice LUTs used as Logic	1,932
	Total equivalent gate count	18,224
	Maximum Frequency	290Mhz
2x2	Slice Registers used Flip-Flops	1,642
	Slice LUTs used as Logic	5,506
	Total equivalent gate count	49,204
	Maximum Frequency	280Mhz

TABLE 3 COMPARISON OF FPGA RESOURCES

	mCGA [9]	**CGA** [10]	**CoCGA normal cell** [10]	**CoCGA leader cell** [10]	**CCGA**
No. of flip-flop	712	541	598	168	621
4-input LUT	1612	1065	1296	359	1932
Total equivalet gate count	18732	12602	17034	4651	18224
Max. frequency	-	330 Mhz	330 Mhz	300 Mhz	300 Mhz

5. Conclusion

In this paper, the CCGA is presented. The results provide initial evidence that CCGA can outperform the normal compact GA and can provide compatible results to the CoCGA with more applicable to FPGA implementation due to unified cell type. The CCGA delivers a more search performance with the adaptive probability vector recombination. For intrinsic evolvable or adaptive hardware, the CCGA can be used for a hardware GA for real-time evolution and adaptation with increased quality of search results. In addition, CCGA can address a scalability issue of genetic algorithm with problem size since CCGA can scale up with problem size by increasing network size as shown in Table 1.

6. References

[1] T. Higuchi, Y. Liu and X. Yao, "Introduction to evolvable hardware", *Evolvable Hardware*, pp. 1-17, Springer 2006.

[2] J. F. Miller and P. Thomson, "Aspects of digital evolution: evolvability and architecture," *Proc. Parallel Problem Solving From Nature*, Amsterdam Netherland, 1998, pp. 927–936.

[3] T. Higuchi et.al, "Real-world applications of analog and digital evolvable hardware," IEEE Transactions on Evolutionary Computation, vol. 3, pp. 220-335, Sept. 1999.

[4] L. Sekanina, "Virtual reconfigurable circuits for real-world applications of evolvable hardware," *Proc. Evolvable systems: from biology to Hardware ICES2003*, 2003, pp. 332–343.

[5] H. Liu, J.F. Miller, and A. M. Tyrrell, "Intrinsic Evolvable Hardware Implementation of a robust biological development model for digital systems," *Proc. NASA/DoD Conference on Evolvable Hardware*, July 2005, pp. 87–92.

[6] P. Haddow and G. Tufte, "An evolvable hardware FPGA for adaptive hardware, "*Proc. IEEE Congress on Evolutionary Computation*, San Diego, CA, 2000, pp. 553–560.

[7] S. Scott and A. Seth, "HGA: A hardware-based genetic algorithm," Proc. ACM/SIGGA 3rd Int. Symp. Field-Programmable Gate Ar-ray,1995, pp. 1-12.

[8] V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for Deep- Submicron FPGAs, Boston, Springer, 1999.

[9] J. C. Gallagher, S. Vigraham, and G. Kramer "A family of compact genetic algorithms for intrinsic Evolvable Hardware," IEEE Transactions on Evolutionary Computation, vol. 8, pp. 111-126, April 2004.

[10] Y. Jewajinda and P. Chongstitvatana, "A cooperative approach to compact genetic algorithm for evolvable hardware," *Proc. IEEE Congress on Evolutionary Computation*, 2006, pp. 624–629.

[11] G. Harik, F. Lobo and D. Goldberg, "The compact Genetic Algorithm*"*, *IEEE Transaction on Evolutionary Computation*, vol. 3, pp. 287-309, Nov. 1999.

[12] E. Cantu-Paz, *Efficient and accurate parallel genetic algorithms*, Boston, MA:Kluwer Academic Publisher, 2000.

[13] C.W. Ahn, D.E. Goldberg, and R. Ramakhrishna, "Multiple-deme parallel estimation of distribution algorithms: basic framework and application. In *Proceedings of Parallel Processing and Applied Mathematics, LNCS 2774,* pp544-551, Springer, 2004

[14] L. DelaOssa et al., "Improving model combination through local search in parallel univariate EDAs," *Proc. IEEE Congress on Evolutionary Computation*, 2006, vol 2, pp. 624–629.

[15] K. Sastry, D.E. Goldberg, and X. Liora "Towards billion-bit optimization via a parallel estimation of distribution algorithm," Proc. GECCO 2004, 2004, pp. 412-413.

[16] M. Sipper, Evolution of parallel cellular machines: the cellular pro-gramming approach, Berlin: Springer-Verlag, 1997.

[17] V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for Deep- Submicron FPGAs, Boston, Springer, 1999.