2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

# A Cooperative Approach to Compact Genetic Algorithm for Evolvable Hardware

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 cooperative compact genetic algorithm (CoCGA). The CoCGA is developed from the compact GA and proposed to be used for intrinsic evolvable hardware. The concept and algorithm of the CoCGA are presented. The hardware implementation of the CoCGA and CGA were carried out. The standard test functions were selected to measure the effectiveness of the CoCGA. The experimental results significantly shows that the CoCGA outperforms the normal compact GA both in speed and quality with acceptable usage of hardware resources for modern-day FPGAs.**

## I. Introduction

Evolvable Hardware (EH) is an emerging research area in Evolutionary Computation (EC). There are a number of methods and techniques that propose to apply the Genetic Algorithm (GA), Evolutionary Strategy (ES), and Genetic Programming (GP) to be implemented in hardware, especially implementation onto FPGAs or other reconfigurable devices [1], [2], [4]. There are two approaches for the design of EH: extrinsic and intrinsic [3], [5]. For extrinsic EH, the evolutionary process is performed off-line. Then the results is downloaded onto the hardware. On the contrary, for the intrinsic EH, the evolutionary process is performed wholly or partly in hardware [6], [7], [8], [9]. However, in order to accomplish the intrinsically on-line evolving in hardware and to utilize hardware resource efficiently, a challenging question is how to modify or invent efficient and improved GA or EA algorithms that can be effectively implemented in hardware.

The trend towards the increasing of density and price per performance of current FPGAs due to advanced semiconductor process technology provides an opportunity for designers and researchers to use larger and faster FPGAs for Evolvable Hardware [11], [12]. With this trend of FPGA technology development, the concept of implementing a group of parallel processing units for EH in a single FPGA chip is feasible [8], [10]. In this paper, the cooperative compact genetic algorithms (CGA) is proposed and it's hardware implementation is explored as the compact GA is one of the key algorithms suitable for hardware implementation [16].

Contrary to the Simple GA (SGA), the compact GA is more suitable for hardware implementation due to using probability vectors [17], [16]. The CGA manipulates the probability vector instead of operating on the actual population. This dramatically reduces a number of bits and memory required to store the population. With this representation, it is practical to use only registers implemented using D-Flip-Flops in digital circuits. Thus, it eliminates the need for Random Access Memory (RAM). The experiment shows that the hardware Compact GA is at least 1000 times faster than a software version [17].

Even though the compact GA has advantage for hardware implementation, but unfortunately, the basic compact GA lacks of sufficient search power for EH applications that requires accuracy and faster processing time. Therefore, the CGA has been improved by adding more techniques like elitism, mutation, and champion resampling [15], [16], [21]. This modified CGA is called *CGA or *CGA family. Recently, MiniPop EA is proposed to be used for EH [24]. The MiniPop EA trades away search-power for the ability to implement the algorithms in small size hardware. However, the *CGA and MiniPop algorithms still perform well on normal EH-control problems. In this paper, the new kind of the compact GA algorithm for EH is proposed called Cooperative Compact Genetic Algorithm (CoCGA).

The CoCGA is designed inspired by the concept of the cooperative genetic algorithm [13]. The CoCGA consists of a group of basic compact GAs that operate cooperatively.

This paper presents the cooperative compact genetic algorithm (CoCGA) that offers higher search power in term of faster searching time and better accuracy of search results with an acceptable utilization of hardware resources, especially suitable for implementing onto today FPGAs that have higher density of hardware resources on a single chip.

This paper begins with a description of basic compact GA in section II. Then, section III introduces models of the parallelized implementation of genetic algorithm in high performance computing systems. In section IV, the Cooperative Compact GA (CoCGA) algorithm is proposed and explained in detail and the pseudo codes is discussed. Section V presents the hardware implementation for CGA and CoCGA. Both hardware circuits are implemented onto an FPGA for performance evaluation and comparison. The benchmarks and the methodology used for measuring the search power of CoCGA are briefly discussed in section VI. Then, the benchmarks results are presented in section VII. Finally, the

0-7803-9487-9/06/$20.00/©2006 IEEE

```
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
```

Fig. 1. Psedocode of Compact Genetic Algorithm

paper concludes with a summary of benchmarks results, the potential and applications of the CoCGA, and discussion of future opened research issues.

## II. Compact Genetic Algorithm

The compact GA(CGA) represents the population as a probability distribution over the set of solutions [14]. 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 section 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 Figure 1.

From the pseudocode of CGA, it is quite straight forward for hardware implementation because each bit of probability vector, p[i], can be updated in parallel. A typical hardware architecture is to design a single bit CGA module for each p[i] [17], [16]. Then, the single bit CGA module is connected together to form a chromosome of a particular length. In addition, the elistism-based compact GA was proposed to improve CGA and the hardware implementation of CGA variants was explored [16], [15], [21].

## III. Parallel genetic algorithm

In order to increase the GA's efficiency, the parallelization of GA has been the active research topics particularly using high performance computer systems [20]. The parallelized GA (PGA) can be categorized into four approaches [20]. These are global, coarse-grained, fine-grained, and hybrid approaches.

*1) Global parallelization:* In this class of model, there is only one group of population. The evaluation of individual and execution of genetic operators are performed in parallel. The evaluation can be parallelized by assigning a group of individuals to a processor node to evaluate and send the results back to the common shared memory or a master node. There is no communication between each processor that evaluates each individual.

*2) Coarse grained parallelization:* This is the popular model for the parallelized GAs. The whole population is partitioned into sub-populations. Within each sub-population, individuals can only mate with others in their own sub-population. However, there is an introduction of migration operator the send some individuals from a local sub-population to other sub-population. There are key parameters for this model: topology, migration rate, and migration interval. The topology defines how each sub-population connects to other sub-populations. The migration rate and migration interval specify how many individuals in each sub-population are migrated and how often they are migrated. In this paper, we initially develop our Cooperative Compact GA based on this model.

*3) Fine grained parallelization:* The whole population is divided further into even smaller sub-population than the coarse grain model. The ideal case is each individual handled by only one processor node. This ideal case rarely happens in real world implementation in high performance computer systems excepting the implementation into special hardware bit-level. In summary, this model is similar to the massively parallel processors

*4) Hybrid parallelization:* This approach is to combine two approaches to solve more difficult problems. For the coarse grained and fine grained approaches, in order to exchange individuals between each sub-population, the question of how to communicate and how costly in term of resources need to be considered. These problems are related to migration parameters: topology, migration interval and migration rate

Since this paper proposes the CoCGA for EH which has fundamental concept in the CGA and the implementation partly resembles the coarse-grained parallel GA, key research projects are explored [19], [18]. The architecture for massively parallelization of the compact GA is proposed by Lobo [19]. The key of the proposed approach is to send the probability vector from each slave processor to the master processor instead of sending a group individual in the sub-population. The experiment is carried on in software using a serial implementation of the proposed parallel compact GA architecture.

In addition, the hybrid parallelization of the CGA is proposed to solve the problem of multi-FPGA partitioning and placement [18]. Like most of research on parallel GAs, the multi-FPGA experiments were performed on a Beowulf cluster.

In the following section, we will explore key important migration parameters namely topology, migration interval

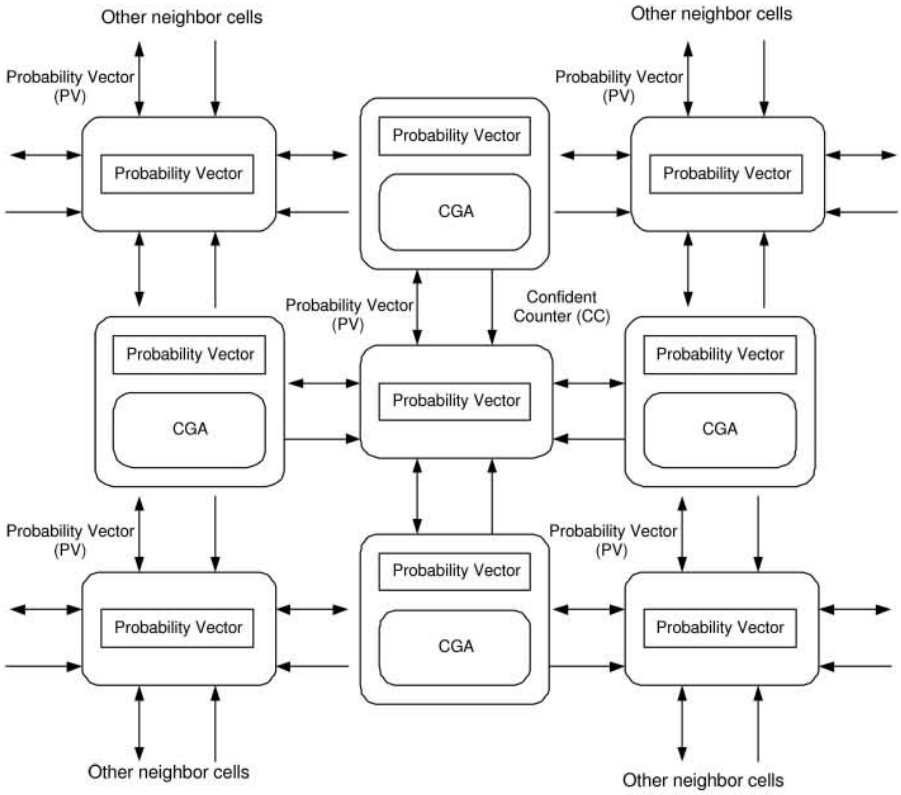


Fig. 2. Cellular Automata (CA)-like topology for Cooperative Compact GA

and migration rate. In the hardware design section, the key parameters are specified quantitatively.

## IV. Cooperative Compact Genetic algorithm

Coevolutionary genetic algorithm applies concept of cooperative approach for function optimization to GA. For cooperative coevolutionary, the search space can be partitioned by splitting the solution vectors into smaller vectors [13], [22], [23]. Each of these smaller search spaces is then searched by separate GAs whose fitness is evaluated by combining solutions found by each of the GAs representing the smaller sub-spaces. In this paper, we presents a cooperative approach for the CGA. The thrust is to propose the concept of *confident counter* that guides toward search direction along with the traditional probability vectors. The cooperation that we proposed comes from using confident counter as a source of shared knowledge on where is the best search direction because only exchanging the probabilities vectors is not enough to guarantee which one of probability vectors from each neighbor cells is the current best vector. In the CoCGA, the individual CGA cell searches in its own sub-population, unlike cooperative coevolutionary approach which split solution vectors into smaller vectors. In our case, the individual CGA cell sends its search result through its probability vector together with its confident counter to the leader cell. The leader cell consolidates this information and decides the bias for the search direction which it then sends back to all neighbor CGA cell as the new current best probability vector.

The implementation of the coevolutionary genetic algorithm is similar to the concept of parallelized GAs described in the previous section. This paper introduces the cooperative compact GA (CoCGA) that uses "shared knowledge" to perform cooperative between many parallel CGAs. Targeting hardware implementation, we propose CoCGA to be applied to the evolvable hardware (EH).

### A. CoCGA topology

Fig. 2 shows the topology of the cooperative compact GA (CoCGA). The topology of the proposed CoCGA resembles the cellular automata (CA) system that cells only interact with their neighbors. However, the interactions between CA cells occur by exchanging the probability vectors instead of mating between individuals of sub-population directly. With this proposed CA topology, the hardware realization of the algorithm is straight forward and not too complicated to be implemented regard to scalability and signal wiring that greatly contribute to the performance of the hardware circuit. In addition, CA architecture has capability of self-evolving and self-replicating [25]. Moreover, CA-like architecture can be practically and efficiently implemented into FPGAs or other reconfigurable devices because of the architecture consists of array of logic blocks [26]. Therefore, CA-like architecture is proposed for the CoCGA.

Each coarse grained CoCGA cell has a probability vector and a sub-population. There is a group leader for each group

```
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
      3.1. Update probability vector
          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

       3.2 Increment     Confidence Counter
          cc: = cc + 1;
  4.  Check if cc is incremented then
       Send p and cc to the group leader cell
  5.  Check if the vector has converged
          for i := 1 to L do
             if p[i] > 0 and p[i] < 1 then
               goto   step 1
  6.  p represents the final solution
 end while
end parallel for
```

Fig. 3. Pseudocode of the normal CoCGA cell

```
L is chromosome length
M is number of neighbor cells
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
        p_l[i] := 0.5;
end parallel for
for each group leader cell       i  in  CA   do in parallel
  while  not done   do
    1.  Check if cc of each neighbor is updated
          if (cc_current[i] <> cc_previous[i])
            goto   step 2
    2.  Select the highest cc of all neighbors
          cc_max := 0;
          for i := 1 to M do
             if (cc[i] > cc_max )
               cc_max := cc[i];
    3.  Update   p_l with p_cc with cc_max
          for i := 1 to L do
             p_l[i] := p_ccmax[i]
    4.  Update new updated p_l to all normal Cell
          for   each neighbor cell  of leader cells
          do in parallel
            for i := 1  to L do
              p[i] := p_l[i];
          end parallel for
    5.  Check if the vector has converged
          for i := 1 to L do
             if p[i] > 0 and p[i] < 1 then
             goto step 1
    6.  p_l represents the final solution
 end while
end parallel for
```

Fig. 4. Pseudocode of the group leader

of these coarse grained CoCGA cells. In Fig. 1, the leader cell is the middle cell that exchanges probability vectors to and from the neighboring cells. The leader cell keeps adjusting its own probability vector to the best probability of the group. The confidence counter (CC) is introduced to help the group leader evaluates which probability vectors from its neighbors are likely to converge to a good solution.

### *B. CoCGA algorithm*

Fig. 3 shows pseudocode of the normal CoCGA cell. After probability vectors of each cell is initialized to the mid-point range, two individuals are generated from the probability vector, then compete "similar to" a normal compact GA. The proposed algorithm is different from the normal compact GA in two ways: (1) the probability vectors are passed to the group leader cells. (2) the confidence toward the better probability vector is calculated as confident counters passed to the group leader cells. In figure 3, the step 3.2 and 4 are inserted into the normal Compact GA.

Fig. 4 gives the pseudocode of the group leader cells which only keep the probability vector but does not implemented the normal compact GA. The group leader updates the probability vectors of its neighbor cells asynchronously because the updating process will occur after the confident counters of the neighbor cells get the new value. For each neighbor cells, the confident counter is incremented asynchronously because its depends on when the current probability vector of each sub-population gives the current best individual. During the search process if the better current individual is found the confident counter is incremented. The group leader will use the probability vector from the neighbor that has the better confident counter. Therefore, the group leader uses an asynchronous updating policy.

The group leader keeps checking if confident counter (cc) for each one of its neighbor are updated to higher value. Once the confident counter for one of its neighbor updated, then the group leader evaluates value of each confident counter and identifies the current highest value in step two. Next,

in step three, the group leader, update its own probability vector with the vector from the neighbor that has the highest confident counter. Then, the new best probability vector is passed from the group leader to its neighbors in step four.

## V. Hardware design

For performance evaluation of the proposed CoCGA in comparison with the normal compact GA in hardware, we designed and implemented two hardware circuits. The first circuit is for the normal compact GA and the second circuit is the proposed CoCGA. Both digital circuits were designed in synthesizable Verilog HDL codes and implemented into an FPGA chip. The CoCGA was designed by adding additional modules to the hardware of the normal CGA hardware. By designing our own hardware for both the CGA and CoCGA, we can perform performance comparison fairly since both hardware circuits designed by the same designers and based on the same Verilog HDL codes.

### A. Hardware Design of the normal CGA

The hardware design for a normal compact GA is similar to the design in [17], [16]. In Fig. 5, the basic unit of the normal CGA is the block named CoCGA bit module/normal CGA bit module without the additional units for CoCGA. For the normal CGA, the hardware design of one bit-module is the same as the CoCGA hardware without additional units.

### B. Hardware Design of the Normal CoCGA Cell

Fig. 6 shows the hardware design of the N-bit module of the two normal CoCGA cells. CoCGA bit-module is based on the design proposed in [17], [16] integrated with the communication unit(COMM) and the confident counter unit(CC). In Fig. 6, the hardware design consists of three main blocks. The first block is the CoCGA bit-module which can be caseaded to form N-bit chromosome. The second block is the additional units for CoCGA which has the confident counter (CC) and the communication unit COMM. The third block is a finite state machine acts as the main controller for the whole block. The detail of these three additional module is described below.

*1) COMM:* COMM is a finite state machine that controls the process of sending and receiving the probability vector as an 8-bit package to and from the normal cells to the lead cell. For a chromosome of N-bit length, the compact GA needs to have N-bit of probability vector which each bit of the probability vector sizes 8-bit. Thus, for N-bit length chromosome, N packages of 8-bit will sent and received between the lead and normal cells by the COMM units of each cell.

*2) CC:* CC is the *confident counter* designed as an 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 lead cell with the current probability vector.

*3) FSM_CONTROL:* FSM_CONTROL is a finite state machine that controls and synchronizes COMM, CC, and the Bit Module.

### C. Hardware Design of the group leader CoCGA Cell

As shown in Fig. 6, the hardware design of the group leader cell is the middle block. The group leader cells consists of four key sub-blocks: two confident counter registers, BestPV, two COMM modules, and Main Controller. The two registers are used to keep the confident counter values from the neighboring cells. Each COMM units handles data to and from each neighboring cells. BestPV is a register that keeps the current best probability vector. Finally, Main Controller is a finite state machine that controls and synchronizes all blocks.

## VI. Benchmark Problems

We used One-Max and the De Jong test functions(F1,F2,F3) to compare the performance of the CGA and the proposed CoCGA. For De Jong's test functions, the solution quality is measured by the objective function value. The function evaluations are performed by modules coded in Verilog-HDL using behavioral modeling with the same precision as described in [27]. For our experiment, the CoCGA has two neighbor cells and one leader cell. The reason to use De Jong's test functions because the De Jong's test function were originally proposed as a means to measure the abilities of search algorithms and used in [14], [16] The functions are fully described in [27]. We used 32-bit chromosome length for One-Max problem. For De Jong test function F1 and F2, we used 30-bit chromosome length. For F3 function, the 50-bit chromosome length was used. The accuracy of the search results were compared to the genesis package [28]. The De Jong's functions F1-F3 are shown below.

De Jong's F1:

$$F1(X) = \sum_{i=1}^{3} x_i^2, \quad -5.12 \le x_i \le 5.12$$

De Jong's F2:

$$F2(X) = 100\,(x_1^2 - x_2)^2 - (1 - x_1)^2, \quad -2.048 \le x_i \le 2.048$$

De Jong's F3:

$$F3(X) = \sum_{i=1}^{5} integer(x_i), \quad -5.12 \le x_i \le 5.12$$

## VII. Experimental Results

In our experiment, the CoCGA consists of two neighboring cells and one leader cell. The hardware implementation of the CoCGA are used to validate the efficacy of the proposed approach. We coded both CGA and CoCGA in Verilog-HDL. The simulation was run on Pentium 4 machine with 512MB memory. The ModelSim Verilog-HDL, an industrial strength simulator, from Mentor Graphics was used to perform the simulation. We evaluated CoCGA by comparing its performance with the performance of a normal CGA on functions: One-Max and De Jong test functions(F1, F2 an F3). The graphs in figure 7 show the best individual,

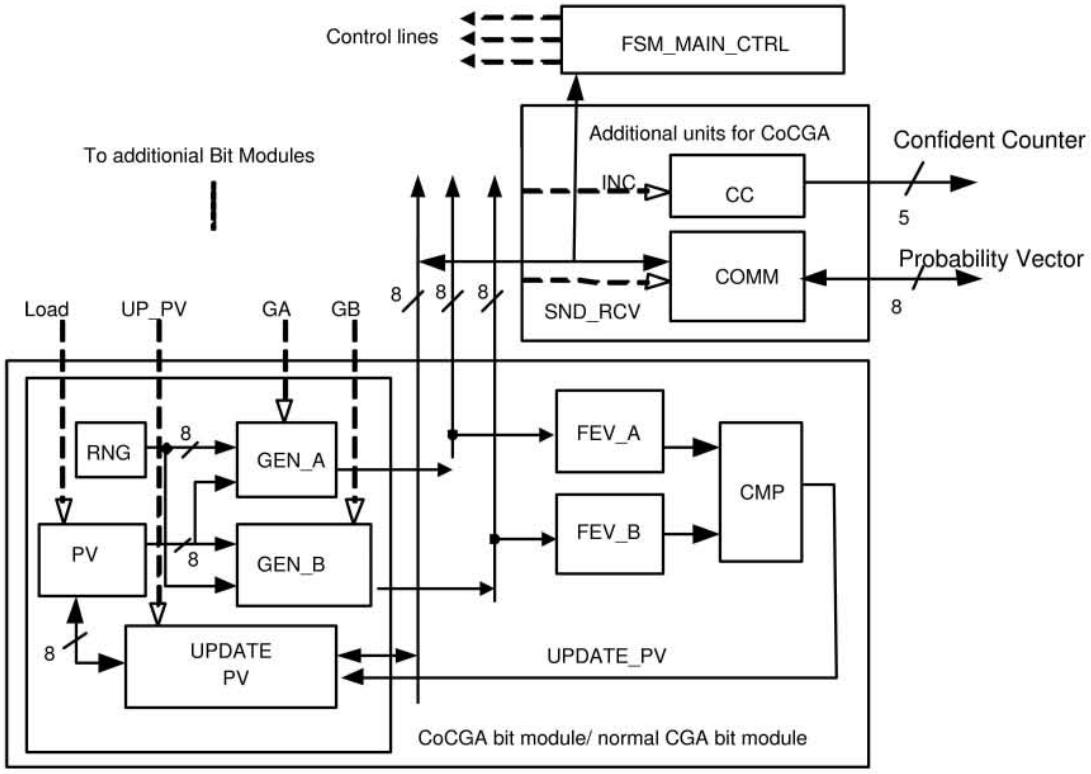


Fig. 5. Hardware block diagram of CoCGA cell

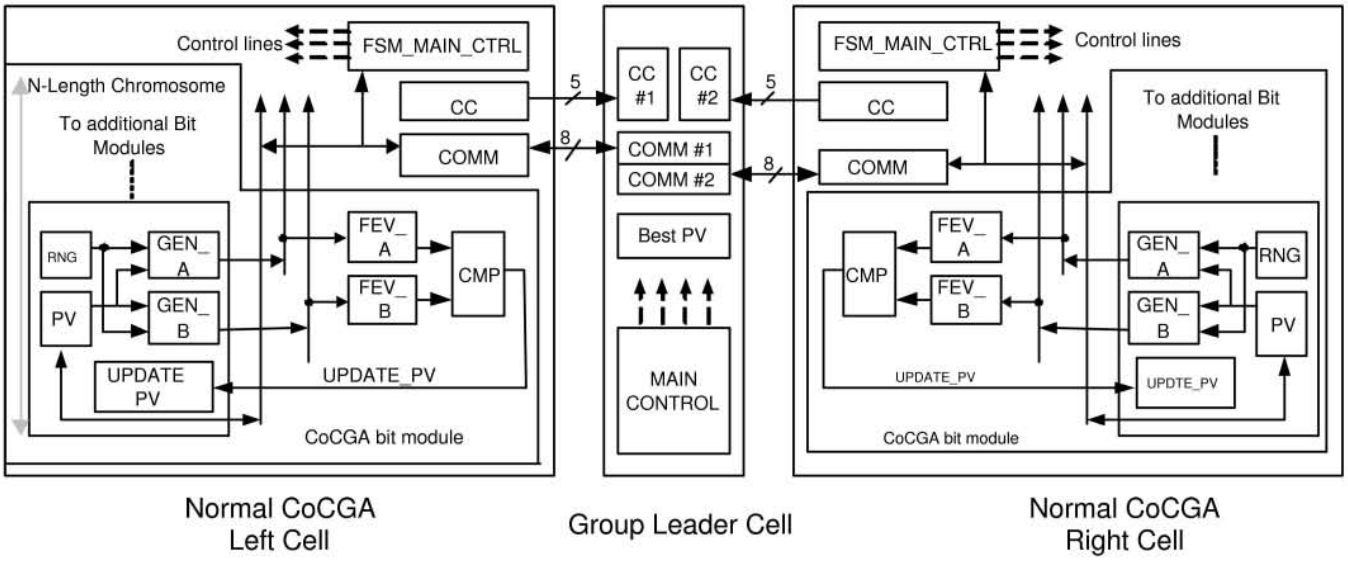


Fig. 6. Hardware block diagram of CoCGA with two neighbors

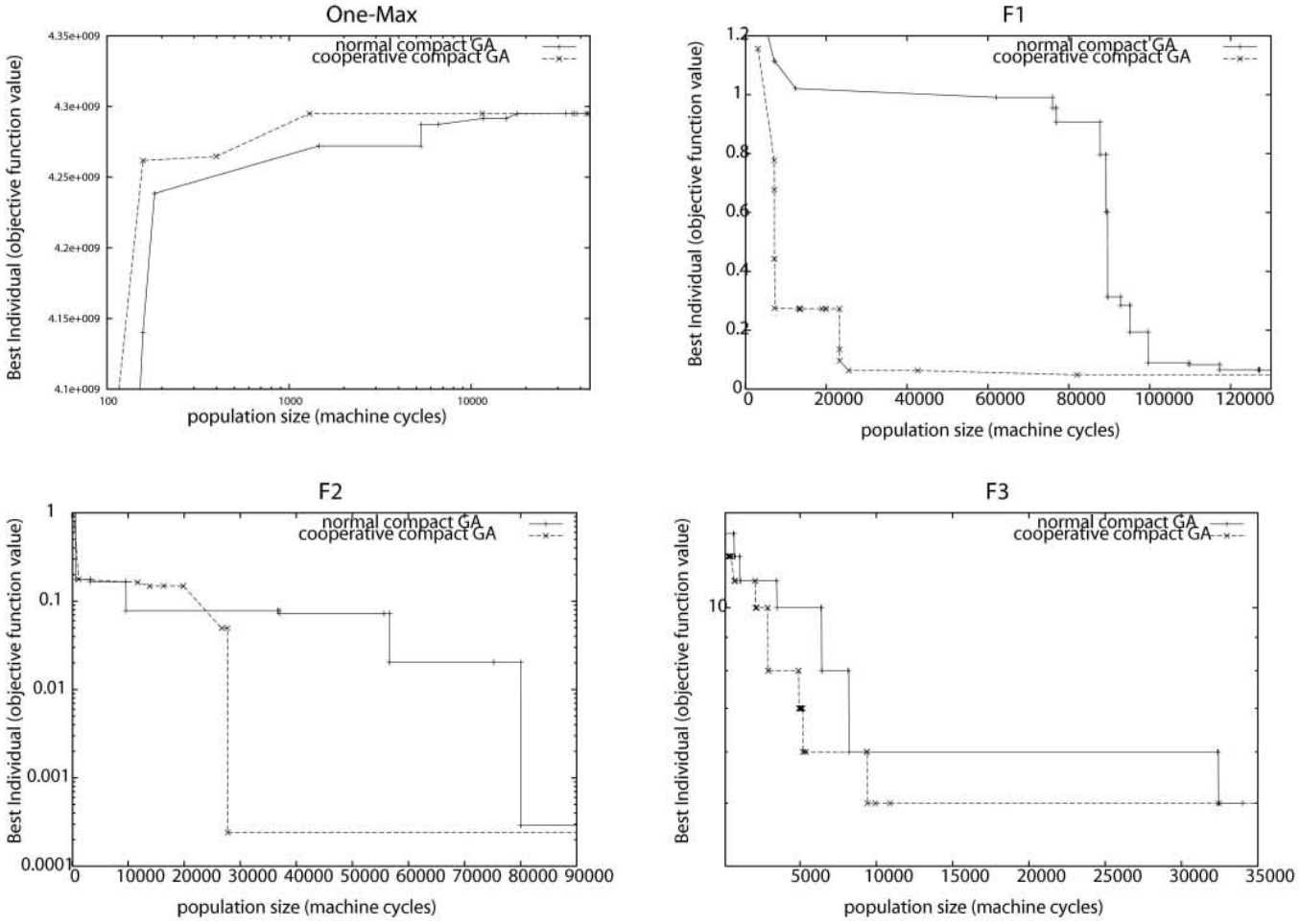


Fig. 7. Comparisons of normal CGA and CoCGA performance

the maximum value for One-Max problem and minimum value for F1-F3. Both algorithms were terminated when each of them converged to minimum values. In all cases CoCGA significantly outperformed the normal compact GA both in the minimum values found and in the speed of convergence. The graph in 7 show statistical significance of the experimental results.

Table I shows the speedup comparison between CGA and CoCGA. From the speedup results, CoCGA outperforms CGA for One-Max, F1, F2, and F3 test functions. CoCGA is at least three times faster than CGA with little higher accuracy for F1 and F2.

We implemented the CGA and CoCGA in Verilog-HDL and synthesized he code into FPGA. For CoCGA, we used one leader cell and two neighbor cells. The verilog design was then synthesized and implemented in to Xilinx Vertex-4 FPGA. Table II shows the FPGA implementation results after the codes synthesized, placed and routed using Xilinx software. From Table II, we can notice that the CoCGA cells only require slightly higher hardware resources than normal CGA cell. This is because we only added the communication unit and confident counter unit to the original implementation of CGA [17], [16].

TABLE I

SPEEDUP COMPARISON BETWEEN CGA AND COCGA IN TERM OF MACHINE CYCLES (ONE MACHINE CYCLE IS EQUIVALENT TO FOUR CLOCK CYCLES)

| | **One-Max** | **F1** | **F2** | **F3** |
|---|---|---|---|---|
| **CGA** | 43362 | 126967 | 80027 | 32427 |
| **CoCGA** | 11492 | 25542 | 27757 | 9407 |
| **Speedup** | 3.77 | 4.97 | 2.88 | 3.44 |

## VIII. CONCLUSIONS

The CoCGA is presented. The results provide initial evidence of the potential of the proposed cooperative compact genetic algorithm in hardware. To make any strong claims concerning its value for Evolvable Hardware, more research on application of the approach is required. For initial application area, the problem of automatically design of digital filters and robot control problems can be experimentally solved using the approach [7], [8], [24]. In addition, several aspects of this approach deserve some attention.

In [7], [8], the intrinsic evolvable hardware which consists

TABLE II
CGA VERSUS COCGA FPGA IMPLEMENTATION

| | CGA | CoCGA normalcell | CoCGA leader cell | mCGA (from [16]) |
|---|---|---|---|---|
| Family | Vertex 4 | Vertex 4 | Vertex 4 | Vertex 2 |
| Device | v1x25 sf363-10 | v1x25 sf363-10 | v1x25 sf363-10 | xc2 v1000 |
| No. of flip flops | 541 | 598 | 168 | 712 |
| 4-input LUT | 1065 | 1296 | 359 | 1612 |
| Total equivalent gate count | 12602 | 17034 | +651 | 18732 |
| Maximum Frequency | 136.325 Mhz | 134.421 Mhz | 145.423 Mhz | 87.187 Mhz |

of a novel reconfigurable architecture based on Cartesian Genetic Programming and a hardware GA is used to implement high performance digital filters. The CoCGA can be used for the hardware GA to provide speedup and to increase quality of the search results.

Gallagher [16], [24] applied and proposed *CGA and minipop algorithms in EH for control applications, especially for robotic controllers. The CoCGA can also be applied to this control applications as well because CoCGA offers higher performance and based on the compact GA like *CGA.

For the aspects that deserved our attention, first, any kind of the compact genetic GA like *CGA and other hardware-based EA can benefit from this approach since they can be used to evolve cooperatively to solve the problems of interests. The approach offers to improve the problem solving capabilities of EH, considering from the nowadays FPGAs and reconfigurable devices technology.

Second, the CoCGA algorithm is a natural mapping onto coarsely grained parallel architectures. So there are opened issues that can be explored in hardware implementation like synchronous and asynchronous update from each subpopulation, and partnering strategies for each neighboring cells.

Finally, the real potential of the cooperative compact GA in hardware will become apparent when applied to specific applications like EH for control applications or automatically design of digital systems [7], [8], [24]. We hope to provide such evidence in the near future.

## REFERENCES


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

[2] T. Kajitai *et al.* "A gate level EHW chip: implementating ga operations and reconfigurable hardware on a single LSI," *Proc. Int. Conf. Evolvable System*, 1998, pp. 1–12.

[3] 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.

[4] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, H. Murakawa, I. Iajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, and N. Otsu, "Real world applications of analog and digital evolvable hardware," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 220–235, Sept. 1999.

[5] 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.

[6] G. Hollingworth, S. Smith, and A.M. Tyrrell, "Safe intrinsic evolution of virtex devices," *Proc. NASA/DoD Conference on Evolvable Hardware*,July 2000, pp. 195–202.

[7] L. Sekanina, "Virtual reconfigurable circuits for real-world applications of evolvable hardware," *Proc. Evolvable systems: from biology to hardwareICES2003*, 2003, pp. 332–343.

[8] Y. Zhang, S. L. Smith, and A.M. Tyrrell, "Digital circuit design using intrinsic evolvable hardware," *Proc. NASA/DoD Conference on Evolvable Hardware*,July 2004, pp. 55–62.

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

[10] A. Hariri, R. Rastegar, K. Navi, M.S Zamani, and M.R. Meybodi, "Cellular learning automata based evolutionary computing (CLA-EC) for intrinsic hardware evolution," *Proc. NASA/DoD Conference on Evolvable Hardware*,July 2005, pp. 294–297 .

[11] K. Bondalapati and V. K. Prasanna, "Reconfigurable computing systems," *Proceeding of IEEE*, vol. 90, pp. 1201–1217, July 2002.

[12] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proceeding of IEEE*, vol. 86, pp. 615–638, April 1998.

[13] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," *Proc. The Third Parallel Problem Solving From Nature*, Jerusalem, Israel, 1994, pp. 249–257.

[14] G. Harik, F. Lobo, and D. Goldberg "The compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 287–309, Nov. 1999.

[15] C. Wook and R.S. Ramakrishna "Elitism-based compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 367–385, Aug. 2003.

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

[17] C. Aporntewan and P. Chongstitvatana, "A hardware implementation of the compact genetic algorithm," *Proc. IEEE Congress on Evolutionary Computation*, Seoul, Korea, 2001, pp. 624–629.

[18] J.I. Hidalgo, M. Prieto, J. Lanchares, R. Baraglia, F. Tirado, and O. Garnica "Hybrid parallelization of a compact genetic algorithm," *Proc. The Eleventh Euro. Conf. on Parallel, Distributed and Network-Based Processing*, 2003.

[19] F. G. Lobo, C. F. Lima, and H. Martires, "An architecture for massive parallelization of the compact genetic algorithm," *Proc. GECCO 2004*, 2004, pp. 412–413.

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

[21] S. A. Vigraham and J. C. Gallagher, "On the relative efficacies of space saving *CGAs for Evolvable Hardware Applications," *Proc. IEEE Congress on Evolutionary Computation*, 2004, pp. 2187–2193.

[22] L. Bull "On coevolutionary genetic algorithms," *Soft Computing 5*, vol. 5, No. 3, pp. 201–207, June 2001.

[23] M. Chang, K. Obkura, K. Ueda, and M. Sugiyama "Modeling coevolutionary genetic algorithms on two-bit landscapes: partnering strategies," *Proc. IEEE Congress on Evolutionary Computation*, 2004, pp. 2349–2356.

[24] G. R. Kramer and J. C. Gallagher "An analysis of the search performance of a mini-population evolutionary algorithm for a robot-locomotion control problem," *Proc. IEEE Congress on Evolutionary Computation*, 2005, pp. 2768–2775.

[25] M. Sipper, *Evolution of parallel cellular machines: the cellular programming approach*, Berlin: Springer Verlag, 1997.

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

[27] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1975.

[28] J. Grefenstette, *Genesis*, Genetic algorithm software in C, 1990.