Proceedings of the 2001 IEEE Congress on Evolutionary Computation Seoul, Korea • May 27-30, 2001

A Hardware Implementation of the Compact Genetic Algorithm

Chatchawit Aporntewan
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Bangkok 10330, Thailand
43718043@chula.ac.th

Prabhas Chongstitvatana
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Bangkok 10330, Thailand
prabhas@chula.ac.th

Abstract- We propose a hardware implementation of the Compact Genetic Algorithm (Compact GA). The design is realized using Verilog HDL, then fabricated on FPGA. Our design, though simple, runs about 1,000 times faster than the software executing on a workstation. An alternative hardware for linkage learning is also proposed in order to enhance the capability of Compact GA to solve highly deceptive problems.

1 Introduction

The Genetic Algorithm (GA) is a powerful optimization algorithm inspired by natural evolution [Goldberg89]. The optimization is performed by creating a population of solutions. In Simple GA, the offspring are produced by standard genetic operators: reproduction, crossover, and mutation. In each generation, a selection scheme is used to select the survivors to the next generation according to their fitness values defined by users. With this artificial evolution, the solutions are gradually improved generation by generation. The GA process starts with a random population and iterates until the termination condition is met (the optimal solution is found, or reaches the maximum number of generations).

Over the past years, GA has been successfully applied to many hard optimization problems. However, the GA process is time-consuming. For many real-world applications, GA can run for days, even when it is executed on a high-performance workstation. Due to the extensive computation of GA, a myriad of hardware-based GAs has been put forward [Scott95, Graham95, Sitkoff95, Bland98, Kajitani98, Yoshida99, Shackleford00]. Here we cite only the more recent works. Most of the cited works present the hardware accelerating the Simple GA, except [Kajitani98, Yoshida99, Shackleford00] that are Steady-state GA. The impressive speedups are depicted in Table 2. However, complex and expensive hardware is employed to attain the speedups. For example, SPLASH2 uses a collection of processor array boards connected to Sun Sparc workstation via an interface card [Graham95]. Another example is the ARMSTRONG, which is a MIMD multicomputer with reconfigurable resources [Sitkoff95]. It consists of an array of processor boards. Each board consists of microprocessor, memory, and FPGAs. Those machines are regarded as reconfigurable computers, developed for general computation. The other works propose custom hardware dedicated for the execution of GA, the implementation still suffers from memory latency limiting the operating clock frequency. The memory bottleneck is inevitable since GA requires a large memory to store the population. As a result, high-speed memory may be used making the hardware expensive, or low-cost memory reducing performance. In contrast to the Simple GA, the Compact GA is more suitable for hardware implementation [Harik99a]. The Compact GA represents a population as an l-dimensional vector, where l is called the chromosome length. The i^{th}-dimension in the vector is a probability of being "0" or "1". Thus the Compact GA manipulates this vector instead of the actual population. This dramatically reduces a number of bits required to store the population. With this representation, it is practical to use *registers* for a probability vector. Consequently, the hardware can execute at the maximum frequency, that is higher than the memory speed. The experiment shows that the hardware Compact GA is 1,000 times faster than a software version.

The remaining sections are organized as follows. Section 2 introduces the Compact GA. Section 3 describes translating the Compact GA into hardware. Section 4 presents the performance evaluation. Section 5 discusses the extension of hardware Compact GA. Section 6 concludes the paper.

2 The Compact Genetic Algorithm

The pseudocode of Compact GA is presented in Figure 1. The Compact GA's parameters are population size (n) and chomosome length (l). A population is represented by an l-dimensional probability vector (p). The $p[i]$ is the probability that the i^{th}-position bit of an individual, randomly picked up from the population, will be one. First, p is initialized to (0.5, 0.5, ..., 0.5). Next, the individual a and b are generated according to p. The fitness values, f_a and f_b, are then assigned to a and b respectively. If $f_a \geq f_b$ then the probability vector will be updated towards the individual a. If $a[i]$ = 1 and $b[i]$ = 0 then $p[i]$ will be increased by $1/n$. If $a[i]$ = 0 and $b[i]$ = 1 then $p[i]$ will be decreased by $1/n$. Note that if $a[i] = b[i]$ then $p[i]$ will not be updated. The loop is repeated until each $p[i]$ becomes zero or one. Finally, p presents the final solution.

3 Hardware Design

It can be seen that the pseudocode of Compact GA is composed of basic operations: add, subtract, and compare. Each

Compact GA parameters:
n: population size.
l: chromosome length.

for i = 1 to l do
  $p[i]$ = 0.5;

repeat
  for i = 1 to l do
    $a[i] = \begin{cases} 1 & \text{with probability } p[i] \\ 0 & \text{otherwise} \end{cases}$
    $b[i] = \begin{cases} 1 & \text{with probability } p[i] \\ 0 & \text{otherwise} \end{cases}$
  endfor

  // Fitness calculation
  f_a = fitness(a)
  f_b = fitness(b)

  for i = 1 to l do
    if $f_a \geq f_b$ then
      if $a[i]$ = 1 and $b[i]$ = 0 then
        $p[i]$ = min(1, $p[i]$ + $\frac{1}{n}$)
      if $a[i]$ = 0 and $b[i]$ = 1 then
        $p[i]$ = max(0, $p[i]$ - $\frac{1}{n}$)
    else
      if $a[i]$ = 1 and $b[i]$ = 0 then
        $p[i]$ = max(0, $p[i]$ - $\frac{1}{n}$)
      if $a[i]$ = 0 and $b[i]$ = 1 then
        $p[i]$ = min(1, $p[i]$ + $\frac{1}{n}$)
    endif
  endfor
until each $p[i] \in \{0,1\}$

Figure 1: Pseudocode of Compact GA.

probability, $p[i]$, can be updated in parallel. In addition, the Compact GA can be partially overlapped. This will allow pipelining that increases the hardware performance. The design of implementation of Compact GA consists of 5 modules: random number generator, probability register, comparator, buffer, and fitness evaluator. The hardware organization is shown in Figure 2.

Random number generator (RNG) A one-dimensional, 2-state cellular automata (CA) is used to produce random numbers [Hortensius89]. Increasing the size of CA yields a better quality of random numbers, however, the 8-bit CA is sufficient for the demonstration. To generate each bit of an individual in parallel, the number of RNGs is identical to the chromosome length.

Probability register (PRB) In Figure 1, the probability, $p[i]$, is a floating-point number. In fact, it can be replaced by an integer representation since the operations performed on $p[i]$ are only add and subtract by $1/n$. Suppose n = 256. Then a 8-bit integer is sufficient for $p[i]$ and the operations performed on the integer are limited to increment and decrement. For that reason, n must be a power of two.

Comparator (CMP) The CMP is a combinational circuit comparing two integers, m and n. If $m \geq n$, then the output will be "1". Otherwise, the output will be "0".

Buffer (BUF) The buffer is a sequential circuit determining the i^{th}-position bits of the individuals "a" and "b". The buffers hold the individuals while they are being evaluated.

Fitness evaluator (FEV) Two evaluators are used to compute the fitness of the individuals "a" and "b" in parallel. For one-max problem, the fitness evaluator simply counts the number of "1" in a binary string. The number of clocks, spent in the fitness evaluation, varies tremendously from toy problems to hard optimization problems.

The hardware Compact GA performs operations on a probability vector, p. Every dimension, $p[i]$, is updated in parallel. The RNG, PRB and CMP units are used to generate two individuals and store them in BUF. The FEV units evaluate the fitness of two individuals. The CMP unit determines the winner/loser and updates the probability vector in the PRBs.

The hardware Compact GA works as follows. When the reset signal is received, the random number generators are seeded with values, the probability registers are set at 0.5, and the buffers are reset to the start state. Next, the following steps are repeated until all probability registers are zero or one.

1. The result of fitness evaluations determines whether an increment or decrement operation is performed on the probability register. Next, the random numbers and the probability registers are compared.

2. The buffers store the comparison result. If the random number is greater than $p[i]$, the i^{th}-position bit of individual "a" will be set to "0". Otherwise, it will be set to "1". While the buffers are clocked, the new random numbers are produced simultaneously.

3. The buffers perform the same operation as in step 2 for individual "b". In this step, the individuals are forwarded to the fitness evaluators, that are combinational circuits. The comparison of the fitness values is used to update the probability registers in step 1.

It is obvious that each step can be executed in one clock. **As a result, the Compact GA executes one generation per three clock cycles for one-max problem.** The number of clocks per generation depends on the optimization problem. For more complicated problems, a generation takes $3 + e$

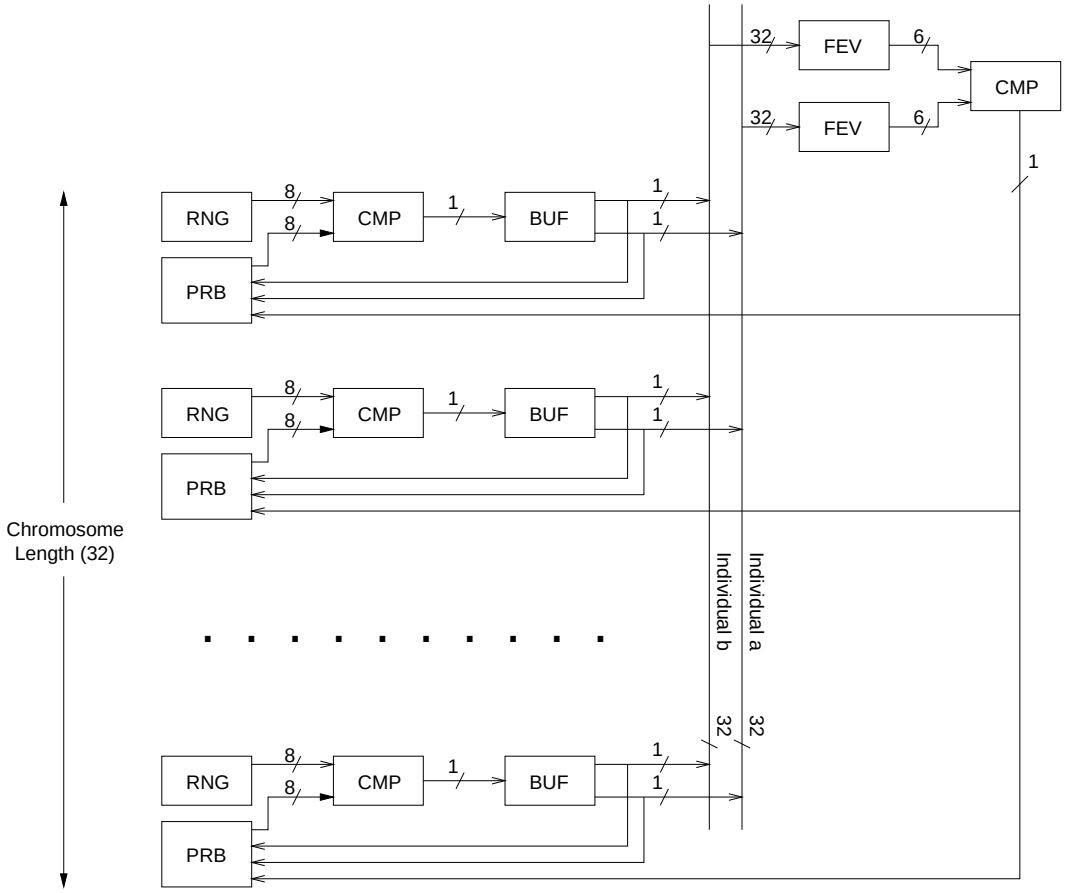

Figure 2: Hardware organization (population size = 256, chromosome length = 32).

Target information:				
Vendor:	Xilinx			
Family:	Virtex			
Device:	V1000FG680			
Speed:	-6			
Design Summary:				
Number of Slices:	813	out of	12,288	6%
Slice Flip Flops:	615			
4 input LUTs:	1,475			
Number of Slices containing unrelated logic:	0	out of	813	0%
Number of bonded IOBs:	2	out of	512	1%
Number of GCLKs:	4	out of	4	100%
Number of GCLKIOBs:	1	out of	4	25%
Total equivalent gate count for design:	15,210			
Additional JTAG gate count for IOBs:	144			
Design statistics:				
Minimum period:	42.423	ns		
Maximum frequency:	23.572	MHz		
Maximum net delay:	10.537	ns		

Figure 3: Synthesis result for one-max problem.

clocks, where e is a number of clocks used in fitness evaluation of an individual. The design is realized using Verilog hardware description language. The population size (n) and the chromosome length (l) are set at 256 and 32 respectively. At the final stage, the design is fabricated on FPGA. The target device is an Xilinx FPGA. The synthesis result for one-max problem is given in Figure 3. The equivalent gate count for the design is 15,210. It is very small. The maximum operating frequency for the design is 23.57 MHz.

4 Performance Evaluation

We choose one-max problem to evaluate the system performance. The population size (n) and the chromosome length (l) are set at 256 and 32 respectively. One million generations are benchmarked. A comparison between software and hardware version is presented in Table 1. The software version is written in C language and compiled using gcc compiler. The software executes on 200 MHz Ultra Sparc II, SunOS. The result shows that the hardware is 1,000 times faster than the software executing on a workstation. The number of generations executed in hardware is $\frac{\text{operating frequency}}{\text{clocks used per one generation}}$ = $\frac{20M}{3}$ = 6.67 million generations per second. It can be seen that the hardware performs much faster.

Table 1: A performance comparison between HW and SW.

Software (200 MHz Ultra Sparc 2)	Hardware (FPGA 20 MHz)	Speedup
2:30 min.	0.15 sec.	1,000

The hardware-based GAs, proposed by others are presented in Table 2. In the last column, the speedups are based on the computers given below. We pick up the best speedups reported in the literatures. To compare our work to the others is difficult since the works use different algorithm, optimization problem, and speedup measurement. The initial works are based on Simple GA [Scott95, Graham95, Sitkoff95]. Due to the reason that Simple GA is not suitable for hardware implementation, the recent works turn to employ the Steady-state GA [Kajitani98, Yoshida99, Shackleford00]. With Steady-state GA, the extreme speedup is achievable. In the latest work, an individual is evaluated every clock cycle [Shackleford00]. We could not claim that the performance of hardware Compact GA is better than the others since the performance evaluations are considerably different. On the other hand, the extreme speedup of the Compact GA over software reveals its ability to capture the benefits of VLSI implementation.

Next, the performance of the hardware Compact GA is generalized to other optimization problems. We divide the software version of Compact GA to two parts: the evaluation part (part one) and the remaining part (part two). Let t_{EV} be $\frac{A}{A+B}$ and let t_{GA} be $\frac{B}{A+B}$, where A is the time spent in part one and B is the time spent in part two. Let s_{EV} be the speedup of hardware over software for part one and let s_{GA} be the speedup of hardware over software for part two. Therefore the speedup of hardware Compact GA for a particular problem can be shown in Equation 1.

$$\text{speedup} = \frac{1}{\frac{t_{GA}}{s_{GA}} + \frac{t_{EV}}{s_{EV}}} \tag{1}$$

The t_{GA}, t_{EV}, and s_{EV} depend on the optimization problem and the fitness function. Moreover, the s_{EV} highly depends on how fast we can evaluate an individual in hardware. The s_{GA} is known since it is problem independent. The s_{GA} can be obtained from Equation 2.

$$s_{GA} = \frac{\text{time spent in part two (SW)}}{\text{time spent in part two (HW)}} \tag{2}$$

Consequently,

$$s_{GA} = \frac{\text{1:48 min.}}{\text{0.10 sec.}} = 1,080$$

In Equation 1, we substitute s_{GA} with 1,080 and substitute t_{GA} with $1 - t_{EV}$.

$$\text{speedup} = \frac{1,080 s_{EV}}{s_{EV}(1 - t_{EV}) + 1,080 t_{EV}} \tag{3}$$

Here the speedup of hardware Compact GA is achieved. It should be noted that the speedup in Equation 3 is based on a 200 MHz Ultra Sparc 2.

The speedup is shown in Figure 4. When the s_{EV} is less than s_{GA}, a problem of which t_{EV} is smaller runs faster. This goes along with the Amdahl's law since the slower part should be as small as possible. When the s_{EV} is greater than s_{GA}, the part two becomes a bottleneck. As a result, a problem of which t_{EV} is larger runs faster.

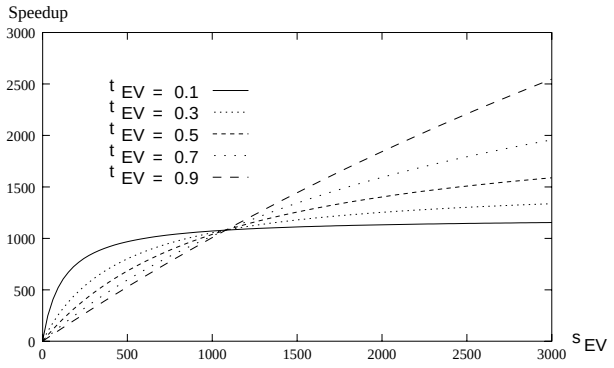

Figure 4: The speedup of hardware Compact GA.

5 Extension

The Compact GA theoretically simulates the oder-one behavior of Simple GA using binary tournament selection and uniform crossover [Harik99a]. Therefore the Compact GA cannot absolutely replace Simple GA for all classes of problems.

Table 2: Performance evaluation.

Developer	Optimization Problem	Performance evaluation
S. Scott (HGA [Scott95])	Linear, quadratic, and cubic functions	18.8X speedup (measured in terms of clock cycles) (Silicon Graphics 4D/440 with four MIPS R3000 CPUs)
P. Graham (SPLASH2 [Graham95])	TSP	10.6X speedup (125 MHz HP PA-RISC workstation)
N. Sitkoff (ARMSTRONG [Sitkoff95])	Chip partitioning	3.0X speedup (60 MHz SPARC Model 61)
I. Kajitani (EHW chip [Kajitani98])	Hand controller	62X speedup (200 MHz Ultra Sparc 2)
N. Yoshida (GAP [Yoshida99])	Data partitioning	NA
B. Shackleford (GA Machine [Shackleford00])	Protein folding	160X speedup (366 MHz Pentium II)

The convergence is ensured for problems consisting of tightly coded, nonoverlapping building blocks. Such problems are rarely found in real-world applications. To enhance the ability to solve highly deceptive problems, *linkage learning* is integrated to the Compact GA [Harik99b]. However, the linkage learning is too complicated for hardware. We have to find another algorithm that performs similar to the linkage learning algorithm but should be simpler to implement in hardware.

An alternative is the *SG-Clans algorithm* [Corno98, Corno99]. The SG-Clans algorithm is similar to Compact GA. The only difference is in generating individual a and b. The SG-Clans algorithm can be shown by inserting the following lines above the fitness calculation in Figure 1.

a = cga(build_clan(a)))
b = cga(build_clan(b)))

After a and b are generated according to the probability vector p, the individual is supposed to be a forefather of a clan. In GA literature, a clan refers to a set of strings which has a common trait (e.g. $\{$1010, 1110, 1011$\}$ belongs to 1*1*). A clan is denoted by a probability vector, p'. The p' is a copy of vector p of which some $p[i]$ are randomly set to "0" or "1" according to the forefather. The `build_clan` procedure returns p' to the `cga` procedure. Next, the `cga` performs Compact GA on p'. Each clan is separately evolved. The `cga` returns the best individual observed during the clan evolution.

We have not yet implemented the SG-Clans algorithm in hardware. However, it can be seen that the hardware for SG-Clans algorithm is straightforward. The `cga` procedure is simply a duplication of hardware Compact GA. In [Corno99], the SG-Clans algorithm is able to find the optimal solution of Holland's Royal Road function (default settings). As a result, the hardware Compact GA could be extended to solve highly deceptive problems.

6 Conclusions

This paper presented a hardware implementation for Compact GA. The hardware Compact GA is simple but effective. The approximate size, including two fitness evaluators, is 15,000 gates. The operating clock frequency on FPGA is 20 MHz. For 32-bit one-max problem, the 1,000X speedup over a software version is achieved. An alternative hardware for linkage learning, adopted from SG-Clans algorithm, is also proposed to solve highly deceptive problems.

7 Acknowledgements

We thank Phillip Rogaway for his helpful comments and a lot of effort contributed to improve this paper. Chatchawit Aporntewan is being supported by Chulalongkorn university's scholarship given in the occasion of the Sixth-Cycle (72^{nd}) Birthday Anniversary Of His Majesty King Bhumibol Adulyadej.

Bibliography

[Bland98] Bland, I. M. and Megson, G. M. "The Systolic Array Genetic Algorithm, An Example of Systolic Arrays as a Reconfigurable Design Methodology," in Proc. of IEEE Symp. on FPGAs for Custom Computing Machines, pp. 260-261, 1998.

[Corno98] Corno, F., Reorda, M. S., and Squillero, G. "A New Evolutionary Algorithm inspired by the Selfish Gene Theory," in Proc. of Int. Conf. on Evolutionary Computation, pp. 575-580, 1998.

[Corno99] Corno, F., Reorda, M. S., and Squillero, G. "Optimizing Deceptive Function with the SG-Clans Algorithm," in Proc. of the Congress on Evolutionary Computation, pp. 2190-2195, 1999.

[Goldberg89] Goldberg, D. E. “Genetic Algorithm in search, optimization and machine learning,” Addison-Wesley, 1989.

[Graham95] Graham, P. and Nelson, B. “A Hardware Genetic Algorithm for the Traveling Salesman Problem on SPLASH 2,” in Proc. of the 5th Int. Workshop on Field Programmable Logic and Applications, pp. 352-361, 1995.

[Harik99a] Harik, G. R., Lobo, F. G., and Goldberg, D. E. “The Compact Genetic Algorithm,” in IEEE Trans. on Evolutionary Computation, Vol. 3, No. 4, pp. 287-297, 1999.

[Harik99b] Harik, G. “Linkage Learning via Probabilistic Modeling in the ECGA,” in IlliGAL Technical Report 99019, 1999.

[Hortensius89] Hortensius, P., McLeod, R., and Card, H. “Parallel Random Number Generation for VLSI Systems using Cellular Automata,” in IEEE Trans. on Computers, Vol. 38, No. 10, pp. 1466-1473, 1989.

[Kajitani98] Kajitani, T., Hoshino, T., Nishikawa, D., Yokoi, H., Nakaya, S., Yamauchi, T., Inuo, T., Kajihara, N., Iwata, M., Keymeulen, D., and Higuchi T. “A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI,” in Proc. of Int. Conf. on Evolvable Systems (ICES’98), pp. 1-12, 1998.

[Scott95] Scott, S. and Seth, A. “HGA: A Hardware-Based Genetic Algorithm,” in Proc. of the ACM/SIGDA Third Int. Symp. on Field-Programmable Gate Arrays, pp. 53-59, 1995.

[Shackleford00] Shackleford, B., Okushi, E., Yasuda, M., Koizumi, H., Seo, K., Iwamoto, T., and Yasuura, H. “An FPGA-based Genetic Algorithm Machine,” in Proc. of the ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays, pp. 218, 2000.

[Sitkoff95] Sitkoff, N., Wazlowski, M., Smith, A., and Silverman, H. “Implementing a Genetic Algorithm on a Parallel Custom Computing Machine,” in Proc. of IEEE Symp. on FPGAs for Custom Computing Machines, pp. 180-187, 1995.

[Yoshida99] Yoshida, N. and Yasuoka, T. “Multi-GAP: Parallel and Distributed Genetic Algorithms in VLSI,” in Proc. of Int. Conf. Systems, Man, and Cybernetics, Vol. 5, pp. 571-576, 1999.