Efficient operator pipelining in a bit serial genetic algorithm engine

Systolic Random Number Generation for Genetic Algorithms

I. M. Bland and G. M. Megson
Parallel, Emergent and Distributed Architectures Laboratory,
(PEDAL),
Algorithm Engineering Research Group,
Dept. of Computer Science,
University of Reading,
Whiteknights,
P.O. Box 225
Reading RG6 6AY.

February 9, 1999

Abstract

A parallel hardware random number generator for use with a VLSI genetic algorithm processing device is proposed. The design uses an systolic array of mixed congruential random number generators. The generators are constantly reseeded with the outputs of the proceeding generators to avoid significant biasing of the randomness of the array which would result in longer times for the algorithm to converge to a solution.

1 Introduction

In recent years there has been a growing interest in developing hardware genetic algorithm devices [1, 2, 3]. A genetic algorithm (GA) is a stochastic search and optimization technique which attempts to capture the power of natural selection by evolving a population of candidate solutions by a process of selection and reproduction [4]. In keeping with the evolutionary analogy, the solutions are called chromosomes with each chromosome containing a number of genes. Chromosomes are commonly simple binary strings (the bits being the genes) and it is these kinds of chromosomes we shall be considering although many other encoding schemes are used (for examples see [5]). Reproduction is performed by two operators, crossover and mutation. Crossover combines random sections of two selected chromosomes to produce a new, hopefully better, chromosome with mutation randomly changing genes to diversify the search. Both crossover and mutation make heavy use of random numbers which, in the case of a hardware genetic algorithm, need to be generated by the device itself.

We have developed a hardware genetic algorithm using a systolic array approach (Fig 1) [6]. The design uses a number of custom systolic arrays to achieve the selection and reproduction of a population of chromosomes. The details are beyond the scope of this letter but the approach exploits massive parallelism and data re-use as well as having uni-directional data flow between cells.

An essential part of the design is a systolic Pseudo-Random Number Generator (PRNG). This generator is capable of producing many streams of random numbers in parallel. To avoid biasing the operators these streams must be sufficiently random and independent from each other. With all algorithmic PRNGs the stream of numbers generated is dependent on the initial seed value. When attempting to construct parallel generators which use the same recurrence relation there is a chance that two generators will be unfortunately seeded and so generate sequences in different cells which have a significant proportion of their numbers in common. In the case a genetic algorithm this would compromise the independence of individual chromosomes and result in longer solution convergence times. Our systolic design overcomes this problem by effectively unrolling the recurrence into the array. Any dependencies introduced as a result of unfortunate seeding are distributed across the array

Figure 1: Block diagram of the systolic array genetic algorithm

in a direction which does not affect the randomness of the generator with respect to itself or with respect to the other random number generators within the array.

2 The Mixed Congruential PRNG

The mixed congruential PRNG is a recurrence relation which takes the form : $X_{n+1} \equiv \lambda_1.X_n + \mu_1(modP)$ and produces a fixed stream of pseudo-random numbers as the equation is iterated. The quality of the random numbers produced depends upon the choice of values for λ_1, μ_1 and P. A maximum period of 2^β can be obtained when $\lambda_1 = 1(mod4)$,$\mu_1 = 1(mod2)$ and $P = 2^\beta$ [7]. An obvious advantage of using this recurrence relation is that it can be constructed in hardware by using only a few registers and adders. The mod P operator can be implicitly applied by using registers of size β (where $P = 2^\beta$). This results in an architecture that is simple and makes efficient use of silicon space.

3 Comparison between other PRNG

Two alternative methods of generating pseudo-random numbers are Linear Feedback Shift Registers (LFSR) and Cellular Automata (CA) based generators . LFSRs generate statistically poor random numbers and rely on communication between non adjacent cells [8]. CA based generators however compare favourably with the mixed congruential PRNG in terms of quality of randomness and use less silicon space in their implementation. To achieve long sequences of pseudo random numbers however, CA generators require a cyclic structure which introduces long communication lines between end cells. This is especially a problem when the PRNG is embedded into the larger design as in our genetic algorithm. CA based methods also require careful seeding to obtain maximal length cycles [8]. Our use of a mixed congruential PRNG requires seeding once and this can be achieved,in the case of a VLSI device, by using the random state the register is in when it is first powered up.

Figure 2: Cell definitions for the generator. (a) Top Cell. (b) Body Cell. (c) Complete systolic PRNG.

4 Implementation

The PRNG is constructed as follows. First we implement a recurrence relation as a systolic array cell, the definition of which is given in Fig2a. This cell generates a constant stream of pseudo-random numbers and passes them out to its neighbouring cell below. We implement a second array cell, which is given in Fig 2b, using *different* values for λ and μ. This cell uses the value passed to it from above as its seed value and passes the result of one iteration to its neighbour below.

The systolic array PRNG is constructed using only these two types of cells as shown in Fig 2c. Chromosomes enter the array staggered, which is consistent with the other genetic operators in Fig 1. The cells possess a simple internal structure consisting of a few registers and adders and can easily be replicated across the design to facilitate scaling of the whole device. In the finished design these cells would be fully incorporated into larger cells which would be responsible for applying the genetic operators to the chromosomes.

5 Usage

To illustrate how the systolic PRNG operates we consider the mutation operator. Assuming the chromosomes are passing through the design in a bit-serial fashion the mutation operator needs to generate N random number per clock cycle (where N is the size of the population). We shall illustrate how independence is preserved inside the array by using an example. Assuming the first recurrence relation produces random number stream A and the second relation produces random number stream B.

Stream A

5, 13, 8, 18, 2, 4, 12, 19, 10, 6, 17, 3, 11, 15, 20, 7, 1, 14, 16, 9.

Stream B

14, 8, 20, 1, 6, 12, 11, 4, 9, 16, 19, 2, 15, 3, 7, 13, 5, 17, 10, 18.

The two streams are chosen so that they will seed the two recurrence relations in such a way as to introduce the maximum dependency between them. Fig 3a gives the contents of the mutation cells as time progresses. Remember the chromosomes are staggered as they enter the array. In the example the two dependent streams

Figure 3: Snapshots of the generator. (a) Numerical values. (b) Mutation decisions

can be seen and have been highlighted. These repeating streams descend through the array as time progresses. As the repeating streams are spread across the array it can be see that the independence of number appearing in the columns (which represent the independence between chromosomes) and the independence of numbers appearing in the rows (which represent the independence of the genes within a single chromosome) is maintained. The random numbers can now be used as a basis for deciding upon mutation. Suppose we choose mutation to occurs if the random number generated by the cell is less than 3. Mutation would occur (indicated by a 'o') as the chromosomes pass through the array as shown in Fig 3b. The mutation probability can also be passed down the array allowing for varying mutation rates. The rate can be altered either on a run by run basis or dynamically as the population begins to converge.

6 Conclusions

By using a pipelined mixed congruential random number generator we have been able to construct a statistically good parallel PRNG using simple computational structures which are locally connected. The design preserves the independence of the random sequences generated by re-seeding each element with the output of the proceeding element after each number has been produced. Dependencies which do occur are therefore distributed across all elements of the generator and therefore do not affect the randomness of any element or the independence of the elements as a whole. We have demonstrated the use of the generator as part of a mutation operator for a hardware genetic algorithm.

References

[1] H. Chan and P. Mazumder. A systolic architecture for high speed hypergraph partitioning using genetic algorithms. In *Lecture Notes in Computer Science*, number 956, pages pp109–126. Springer-Verlag, 1995.

[2] P. Graham and B. Nelson. A hardware genetic algorithm for the travelling salesman problem on SPLASH 2. In *Lecture Notes in Computer Science*, number 957, pages pp352–361. Springer-Verlag, 1995.

[3] B. C. H. Turton and T. Arslan. A parallel genetic VLSI architecture for combinatorial real-time applications - disc scheduling. In *Proc. first IEE/IEEE Int. Conf. Genetic Algorithms in Engineering Systems : Innovations and Applications*, pages pp493–498, Sept. 1995.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[5] Lawrence Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[6] G. M. Megson and I. M. Bland. A generic systolic array for genetic algorithms. Technical report, Dept. of Computer Science, University of Reading, U.K., 1996.

[7] B. Jansson. *Random Number Generators.* Victor Pettersons Bokindustri Aktiebolag, Stockholm, 1966.

[8] P. D. Hortensius, H. C. Card, and R. D. McLeod. Parallel random number generation for VLSI using cellular automata. *IEEE Trans. Computers*, vol. 38:pp 1466–1473, Oct. 1989.