Parallel Implementation of Genetic and Adaptive Partitioned Random Search Algorithms

J R Raol, Nonmember
A M Jalisatgi, Non-member
J Jose, Non-member

Some novel parallel schemes on genetic and adaptive partitioned random search algorithms are presented. They are realised on Flowsolver - NAL's parallel computer platform using C language. Performance studies and implementation aspects are discussed.

Keywords: Algorithms; Parallel implementation; Random search: C-language; Genetic/Adaptive

INTRODUCTION

In recent years the Genetic Algorithms (GAs) are being studied for their applications to optimisation problems. GAs are based on the mechanics of natural selection and genetics. Acting on the premise of survival of the fittest, a population of possible solutions is combined in a manner, similar to mixing of chromosomes in a natural genetic system. The fitter population members pass on their structures as 'genes' in far greater quantities than weaker solutions. The net effect is evolution of the population towards an optimum. The genetic algorithm is a search procedure, which uses random choice as a tool to guide a highly exploitative search through a coding of a parameter space. Due to the search being carried out simultaneously with many samples of the variables over several generations to obtain robust solutions, the computational burden will be very high. Hence, it is necessary to use parallel computers to obtain fast solutions. In this paper, such schemes are developed and implemented on NAL's parallel computers.

FEATURES OF GENETICALGORITHMS

GAs search from a population of points and use only an objective function information. GAs use random transition rules to guide the search, but are not random search methods.

Chromosomes

They represent an encoding of information upon which a GA operates. Each chromosome consists of string of bits (binary 0 or 1) from a set of more than two elements.

Population and Fitness

GA operates by maintaining a population of chromosomes. The population members are called individuals. Each individual is assigned a fitness based on object function value (FV).

Reproduction/Crossover/Mutation

In this process, individual strings are copied according to their objective the next generation. Ment strings in a pool are mated at random. Each pair of strings undergoes a crossover which randomly selects a site along the length of chromosome, and then splits each chromosome into two pieces at the site. The new chromosomes are formed by mating the top piece of one chromosome with the bottom piece of another at a crossover site. This operation efficiently builds new ideas from the best partial solutions of previous trials. Mutation is the random alteration of a bit in string position.

For the problems in science and engineering, the real numbers can be directly used with some modifications in crossover and mutation. The crossover may involve only averaging of the information. Mutation is also simple, in which a small noise is added to the information.

NAL's PARALLEL COMPUTER-FLOWSOLVER

The Flowsolver Mk-3 is based on Intel 860 RISC microprocessors has eight processors in distributed memory message-passing configuration, with 640 MFLOPS of peak computing power. Parallelizing codes on the Flowsolver Mk3 is entirely user-driven, with the inter processor communication being handled by subroutines in the Concurrent Executive (CCX) Library—a system software package. The availability of MPC (message passing coprocessor) and DMA (direct memory access) chips on each processor board frees the CPU from most of the data transfer exercise. The CPU acts as the HOST processor (HP), whereas the other processing elements (PEs) run program as loaded on to them by the HOST which can also take part in the computations. The HOST can run multiple programs at the same time, whereas the PEs are limited to one program at a time. The HP runs under the UNIX operating system and the PEs run under the operating system Kernel called the Monitor. The HOST acts as the user-interface to the machine and to all its CPUs. AU disk I/O is handled through the HP. PEs route their data through the HOST for accessing disk. This is achieved easily by a set of CCX library subroutines. PEs have limited console I/O facility that may be used for debugging purposes at mn time.

PARALLELISATION SCHEMES OF GAs

Since GAs can work on population of points simultaneously, their natural parallelism can make use of parallel processing. In GAs which work with floating point numbers, mutation and crossover routines are independent of each other and therefore these two can work in parallel. And similarly, the routine, which selects new samples, is also independent of other genetic processes in a particular generation, hence can be parallelized. A simple parallel algorithm which is implemented on two processors, can be developed by parallelizing...
Crossover, mutation, selection of new samples and sorting operations.

**Scheme 1**

The processor HP and one PE work simultaneously. HP works on initialisation and initial sorting, then crossover, creating best individual and sorting all the individuals. The PE works on the job of selecting new samples and mutation.

**Host Processor**

1. Select $N$ samples and compute their fitness.
2. Sort initial samples as per their fitness values.
3. Send best individual to the PE for mutation.
4. Crossover $(N + 1)/2$ best samples to generate $(N - 1)/2$ new individuals and calculate their fitness.
5. Carry the best member of previous population to the new population.
6. Receive $(N - 1)/2$ mutated individuals and $N$ new samples from the PE.
7. Create the best individual by combining all individuals, and insert it into the population.
8. Sort the whole population according to FVs.
9. Repeat the steps 2 to 8 for given number of generations.

**Processing Element**

1. Select $N$ samples from the search space/compute the fitness.
2. Receive the best individual from the host for mutation.
3. Generate $(N - 1)/2$ offsprings by mutating the best individual, and compute their fitnesses.
4. Send $(N - 1)/2$ mutated samples and $N$ new samples to the HP.
5. Repeat steps 1 to 4 for equal number of generations as host is doing.

**Example 1**

**Objective:** Minimisation (global)

Function: $f(x, y) = x^2 + y^2 - 5x - 3y + 10$

where $-10.0 \leq x \leq 10.0$ and $-10.0 \leq y \leq 10.0$

Global Max: $f(x, y) = 290.0$ at $x = -10.0$ and $y = -10.0$

Global Min: $f(x, y) = 1.5$ at $x = 2.5$ and $y = 1.5$

Constant of mutation: $0.125$

Results show that the efficiency of parallel algorithm is as good as sequential algorithm. Figs 1 and 2 show the comparisons of performance. Accuracy is the difference between the actual and the computed value of the cost function at global optimum. Times taken by various routines for a particular problem are shown in Fig 3. In Scheme 1, crossover and mutation were parallelised. But time taken by crossover was 0.5 ms, whereas that of mutation is 1.8 ms. Hence, most of the time host waits for PE to send mutated individuals. Therefore, net time taken is 1.8 ms. Selection of new samples and calculation of their fitness, was parallelised with other processes such as sorting and creation of best individual. But here selection of new samples takes 3 ms whereas sorting takes 20 ms and routine for creation takes 0.5 ms. This time, PE waits for the host to send the best individual for mutation. For this part net time taken is 20.5 ms and time saved here is only 3 ms. By parallelising, time saved together is only 3.5 ms. But each inter processor data transfer consumes a time of 3.5 ms. With two such data transfers, total time taken by inter processor data transfer was 7 ms. Therefore net effect of parallelization was losing 3.5 ms in each iteration.

**Scheme 2**

An improvement over Scheme 1, can be achieved by parallelising only such routines which take equal times, and decreasing the number of inter processor communication. By carrying out mutation in the HP, one inter processor data transfer can be saved. Since sorting is the largest time consuming routine it can be split between both the processors, and then combined with one more small routine which takes only 0.5 ms. In the new scheme, HP does the job of crossover, mutation, create, sorting the crossover and mutated individuals and finally, merging with the individuals, sent by the PE which select new samples and sorts these new samples according to fitness (Fig 4).

**Host Processor**

1. Generate initial population and compute their fitness.
The performance remains as good as in sequential execution, however, the time taken by parallel GA was 8.21 s as against one processor which took 12.63 s.

**Scheme 3**

In this scheme, Fig 5, no specific work is assigned to any processor. Instead of dividing the process among the processors, the total cycles are divided among them. The processors exchange the best value among them at regular intervals. The number of generations should be a multiple of the number of processors. There is no difference between the HOST and the PEs except that the I/O is done through the HOST—the programs for HOST and PEs are the same. The cycle for exchange is decided by the number of times the data is exchanged. The value is to be selected according to the speed and desired level of performance. The more number of times the data is exchanged, the better would be the performance and slower would be the process. At the host side provisions for I/O is to be made. This method has the advantage that, it can be used with any number of processors, without much change in the code.

**Example 2**

It is the same as example 1. The performance in terms of percentage success by the sequential and parallel executions were slightly different. Here in this scheme, the cycles in each generations are parallelised, i.e., the total number of generations is shared by the processors. Due to the difference in random numbers generated, one processor may get a better value than the others for the same objective function. When the population is large, convergence is reached earlier. The time taken (for 201 samples; 20 generations; 1000 trials, accuracy 0.0001) for sequential and parallel algorithms are 206 s and 125 s, respectively. The effect of number of samples for a given number of generations and effect of number of

**Processing Element**

1. Select N samples and compute the fitness.
2. Sort the selected individuals according to their fitness.
3. Send N new sorted samples to the host processor.
4. Repeat steps one to three for equal number generations as host is doing.

**Fig 5 Schematic of parallel GA of Scheme 3**
A fixed number of function evaluations (referred to as the regional sample size) are performed in each of \( K \) rectangles using the IUD (independent and uniformly distributed) sampling scheme.

Based on the sampled function values, the promising index of each of the \( K \) rectangles is estimated and the most promising sub-region is determined.

The most promising rectangle is further equally partitioned into \( K \) rectangles and the remaining un-partitioned rectangles are aggregated into a single SR referred to as the surrounding sub-region, Fig 6(b).

Again a fixed number of function evaluations are performed in each of the \((K_0 + 1)\) sub regions \((K_0\) rectangles and one surrounding sub-region) and then promising indices are estimated.

The process is repeated until the size of the most promising sub-region is smaller than some pre-specified number, which is the measure of the length of the longest dimension of the most promising sub-region.

**Implementation of APRS**

APRS is implemented on one processor in Fig 7. Here the best function value is taken as the criterion for determining the most promising index. The best value is taken from each of the sub-regions and the sub-region with the highest value is taken as the best region. This is suitable for the implementation and the computation overhead per cycle is reduced. Even though this method requires a few more function evaluations than the original APRS, the total time may not vary much.

Secondly, instead of taking a region, the \( X \) and \( Y \) boundaries of the region are considered separately. This is suitable for the generation of random samples in each of the regions. This helps in the handling of functions with higher dimensions. In one cycle function evaluation for each variable is done separately, ie, at that time the other variables would be kept as constants. This value is taken for the function evaluation for the next variable and thereby a considerable amount of accuracy is obtained in each cycle. So after each cycle all the variables would be improved in their values. If the best area happens to be the surrounding sub-region, its further subdivision becomes difficult. To solve this problem, initially the entire sub-region in each dimension is divided into a number of sub-regions and their boundary values are stored. If the most promising sub-region turns to be the surrounding region, then this array is searched to find the region which contains the best value. Then the process is continued by dividing this region. In the original concept, the entire surrounding region is to be subdivided. This method minimises the area of the surrounding region to be subdivided. The same algorithm can be used for the determination of the global optimum of functions with more than one variable or dimension. In that case the algorithm is applied separately for each dimension with the best available values of the rest of the variables. If the function has two variables, say \( X \) and \( Y \), first some arbitrary values are assigned as the best values of \( X \) and \( Y \). Then the best value of \( X \) is found out taking the best \( Y \) value. Then the best value of \( Y \) is calculated using the new best value of \( X \). This increases the speed of convergence. Here, for the measurement of accuracy and percentage of success, the number
COMBINATION OF APRS AND GA

This APRS-GA combination is basically an APRS method. GA is used in the algorithm, for estimating the most optimal value in a sub-region. The performance varies in the same way as that in the APRS, with varying parameters. In the GA part, the number of generations can improve the overall performance of the algorithm. The steps involved in the algorithm are:

1. Divide the search space into \((K - 1)\) equal parts.
2. Using GA, find the most optimal value in each SR and determine the best SR based on these values.
3. Divide the best sub-region into \((K - 1)\) equal parts and aggregate the rest of the regions as the surrounding region.
4. Again use GA to find the best sub-region and repeat this process until desired accuracy is reached.

This algorithm has been successfully implemented on NAL's Flosolver MK3. In the GA part of the example 3, the number of generations is 20. For this study, number of trials were 100 (accuracy = 0.0001). The percentage of success in obtaining global maximum and minimum for this case was nearly 100 for number of partitions and samples greater than 5. It is clear that the APRS and GA have better performance than the APRS alone.

PARALLELISATION OF NEW COMBINED ALGORITHM

Although the combined algorithm is more efficient than the APRS, the computational overhead is large. APRS method by itself is fast, but when combined with GA it becomes slow. So it can be parallelised efficiently by assigning a sub-region or a group of sub-regions to different processors. Since APRS goes through only a limited number of iterations, this makes the algorithm faster. In this case, the inter-processor commun-

---

**Example 3**

Global maximisation/minimisation of fun:

\[- \sin (x) + \cos (y), \text{ where } 0 < x < 360 \text{ and } 0 < y < 360;\]

\[(- \sin (x) + \cos (y))/2, \text{ where } 360 < x < 500 \text{ and } 360 < y < 500 \]

Global Max : \(f (x, y) = 2.0 \text{ at } x = 270 \) and \(y = 0\)

Local Max : \(f (x, y) = 1.0 \text{ at } x = 270 \text{ and } y = 360\)

Global Min : \(f (x, y) = - 2.0 \text{ at } x = 90 \) and \(y = 180\)

Local Min : \(f (x, y) = - 1.0 \text{ at } x = 450 \) and \(y = 180\)

The results are shown in Fig 8. The performance suddenly falls when the number of partitions is 12.

---

**Fig 7 Schematic of APRS method**

**Fig 8 Effect of number of partitions on percentage success (number of iterations : 9; number of samples : 9; accuracy : 0.0001; number of trials : 1000)**

**Fig 9 Schematic of APRS-GA combined algorithm Scheme 4**
CONCLUSION

Several new parallelisation schemes of GA/APRS have been presented and evaluated. Useful performance results have been demonstrated by implementing the schemes on NAL’s parallel computer platform for some representative optimisation examples which handle real numbers.

ACKNOWLEDGMENTS

The authors are grateful to Dr U N Sinha, Head, Flosolver Group, NAL and his colleagues for their support during the course of this work.

REFERENCES


Statement about ownership and other particulars about the Journal of the Institution of Engineers (India)
(To be published on the first issue every year after the last day of February)

FORM IV
(See Rule 8 of the Registration of Newspapers) (Central Rules 1956)

1. Place of publication 8 Gokhale Road, Calcutta 700 020
2. Periodicity of its publication Monthly
3. Printer’s Name B C Thakurta
   Nationality Indian
   Address 8 Gokhale Road Calcutta 700 020
4. Editor’s Name B C Thakurta
   Nationality Indian
   Address 8 Gokhale Road, Calcutta 700 020
5. Names and addresses of individuals who own the newspaper and partners of shareholders holding more than one per cent of capital The Institution of Engineers (India)
   8 Gokhale Road, Calcutta 700 020

I, B C Thakurta, hereby declare that the particulars given above are true to the best of my knowledge and belief.

Dated: March 1, 2000

Sd/- B C Thakurta
Signature of Publisher