Genetic Algorithm: Simple to Parallel Implementation using MapReduce

Girdhar Gopal
Research Scholar
DCSA, KUK
Haryana, India

Rakesh Kumar
Professor
DCSA, KUK
Haryana, India

Naveen Kumar
Research Scholar
DCSA, HPU
Shimla, India

ABSTRACT

Simple Genetic Algorithms are used to solve optimization problems. Genetic Algorithm also comes with a parallel implementation as Parallel Genetic Algorithm (PGA). PGA can be used to reduce the execution time of SGA and also to solve larger size instances of problems. In this paper, different implementations for PGA have been discussed with their frameworks. In this implementation, all PGA are based on a single SGA framework. These are executed on a parallel machine and tested on some benchmark problem instances of Traveling Salesman problem (TSP) from TSPLIB. TSPLIB is a well known library for data set of benchmark problem instances. A basic framework has been proposed for implementing PGA on today's parallel computers.

General Terms

Genetic Algorithm, Parallel Genetic Algorithm, Traveling Salesman Problem

Keywords

Optimization, Parallel Genetic Algorithm, Simple Genetic Algorithm, Traveling Salesman Problem

1. INTRODUCTION

Genetic Algorithm is a population based meta-heuristic search technique consisting of following operators: Initialization, Selection, Reproduction and Replacement [1] [2] [3] [4]. Firstly, the population is generated randomly to begin with. Then selection operator finds the fittest members. This operator follows the "survival of the fittest" principle. These fit members are used in reproduction to create new offspring's. Crossover and Mutation are used in reproduction phase. Crossover creates new child from parents and mutation is used to alter the chromosomes to remove the problems of genetic drift etc. Finally replacement is used to manage the old and new set of chromosomes to iterate for further generations. GA is good to arrive at basins of attraction in a large solution space. And to reduce the large amount of computation time PGA can be used.

Traveling Salesman Problem is a routing problem with many possible solutions. But the problem must adhere the optimality of solutions. This optimality leads to the curse of dimensionality. It is a well known NP-Hard combinatorial optimization problem [5]. It can be described as a collection of N number of cities and one salesman, which needs to visit every city exactly once and return to the starting city from the collection of cities. The goal is to find the optimal/minimum cost path. The number of solutions in an N-city problem will vary from (N-1)! To N!, which becomes worse soon with large values of N.

This paper is organized in following sections. Section 2, details about Simple Genetic Algorithm is provided. In section 3, Parallel Genetic Algorithm has been discussed. The implementation details of SGA and PGA are provided in section 4. Finally, section 5 presents the conclusions and findings of the paper.

2. SIMPLE GENETIC ALGORITHM

GA is an adaptive and heuristic based search technique which can be used to search from the search space. GA is proposed by John Holland [1] at University of Michigan in 1975. These are also described as adaptive heuristic search algorithms [2] based on the evolutionary ideas of natural selection and natural genetics by David Goldberg. GA makes progress toward the optimal solution by combining better and better solutions in each generation to create more better solutions [5] [7]. Each single solution is represented by chromosomes, which will be evaluated for the fitness of that solution. This fitness is used as a guide about the solution quality. This process continues to achieve the optimal solution. General structure of genetic algorithm is:

```
Procedure SGA (fitness, pop_size, pc, pm,
no_of_generations)

// fitness is the fitness function used to evaluate
chromosomes in population

// pop_size is the population size in each generation (say
500)

// pc is the probability of crossover (say 0.90)

// pm is the mutation rate (say 0.001)

// no_of_generations is total number of generations

        pop = generate pop_size individuals randomly to
start with

        gen_number =1               //denotes current
generation

        while gen_number <= no_of_generations do {

                L = Select(pop, pop_size, nogen)

                S = Crossover(L, pop_size, pc)

                M = Mutation(S, pop_size, pm)

                pop = M

                gen_number = gen_number + 1;

        }

end proc
```

3. PARALLEL GENETIC ALGORITHM

It is hard in general to list the parallel computer architecture, as it has a very large history. Starting from Pipelining to Vector architectures, from Dual core machines to current i3, i5, i7 multi-core architectures and from single CPU to distributed systems and cloud computers. There are two kinds of parallel GAs that can exploit these kinds of parallel architectures very effectively: multiple-population GAs (also called coarse grained or island model GAs) and master-slave (or global) parallel GAs.

GA is very complex algorithm that is controlled by many parameters and its success depends largely on setting these parameters adequately. The problem is that no single set of parameter values will result in an effective and efficient algorithm in all solutions [3] [6]. For this reason, the fine tuning of a GA to a particular application is still an art and science. Master-slave GA is a simple GA that distributes the evaluation of the population among several processors. Whereas island-model GA each processor starts with an independent population.

4. IMPLEMENTATION & RESULTS

All of the algorithms are implemented on an IBM Xeon Dual Core Server with 60GB RAM and 1TB Hard Disk. SGA has been implemented as the algorithm stated in section 2. Various functions of GA are used as below for implementing it to optimize TSP.

- Population Size: 1000
- Initialization: Random
- Selection: Roulette-Wheel Selection
- Crossover: Partial Matched Crossover with 0.9 probabilities.
- Mutation: Swapping with 0.01 probabilities.
- Replacement: Simple replacement (λ, μ).
- Termination: When the best solution not improved for 100 generations.

Parallel Genetic Algorithms are implemented using the parallel computing toolbox of MATLAB. Parallel Computing Toolbox™ allows the sharing of work within MATLAB clients. A number of clients can be made with the help of *matlabpool* command. This pool can exploit the number of cores in a processor and hyper threading also. It will create as many clients of MATLAB execution engine as there are virtual processors. These multiple workers can be used to do multiple tasks at the same time. A local worker can be used to keep MATLAB client session free for interactive work, or with MATLAB Distributed Computing Server one can take advantage of another computer's speed [8] [9] [10]. Also the MATLAB Distributed Computing Server allows running as many MATLAB workers on a remote cluster of computers as the licensing of MATLAB allows. In this implementation, the main *for* loop of generations is replaced with the *parfor* loop provided by the MATLAB toolbox. To interactively run code that contains a parallel loop, firstly MATLAB pool must be opened. It reserves a collection of workers to run the loop in parallel. The MATLAB pool can consist of MATLAB sessions running on local machine or on a remote cluster:

```
matlabpool open local 3
```

It will open a pool of three workers in the machine. When you are finished with your code, close the MATLAB pool and release the workers:

```
matlabpool close
```

Secondly PGA is implemented as MapReduce framework. MapReduce is a programming technique for analyzing data sets that do not fit in memory. MapReduce is a programming model to process large datasets and make use of computing resources of each server's CPU. It comprises of two phases: Map phase and Reduce phase. In Map phase Mapper must be able to ingest the input and process that input record and then that processed record is forwarded to Reduce phase, where task will be reduced. The Mapper takes in a key/value pair and generates intermediate key/value pairs [11]. The reducer merges all the pairs associated with the same intermediate key and produce the final output that is list of key/values. Every job must contain one map function followed by optional reduce function, these steps need to follow this certain order.

Large instances of TSP generate lots of data in between the solution. MATLAB® provides an implementation of the MapReduce technique with the *mapreduce* function. This implementation is however slightly different from Hadoop MapReduce. *mapreduce* uses a special abstract data type called *datastore*. It is used to process data in small chunks which fits in computer RAM. Each of these chunks goes through the map and reduce functions. These functions are available as *map* and *reduce* in MATLAB. To solve any problem with mapreduce, one has to write these two functions for the data and then these are passed as inputs to the main *mapreduce* function. There are endless combinations of map and reduce functions to process data, so this technique is both flexible and extremely powerful for tackling large data processing tasks.

To implement MapReduce functionality in MATLAB for this experiment, following steps are followed: -

- Data Preparation: Firstly the population is evaluated for fitness.
- Map and Reduce Functions: Selection operator finds the best individuals for reproduction. The inputs to the map function are `data` and `intermKVStore`. Where `data` is the result of a call to the `read` function on the input `datastore`. `intermKVStore` is the name of the intermediate Key-Value-Store object to which the map function needs to add key-value pairs.
- Run MapReduce: After having a *datastore*, a *map* function, and a *reduce* function, *mapreduce* function is called to perform the calculation. i.e.

  ```
  outpop = mapreduce(ds, @TSPMapFun, @TSPReduceFun);
  ```

  ```
  ********************************
  *     MAP - REDUCE PROGRESS    *
  ********************************
  Map   0% Reduce   0%
  Map  14% Reduce   0%
  Map  34% Reduce   0%
  ```

Map 58% Reduce 0%

Map 78% Reduce 0%

Map 96% Reduce 0%

Map 100% Reduce 100%

- View Results: `readall` function can be used to read the key-value pairs from the output *datastore*. i.e. *readall(outpop)*

After performing these steps the resultant chromosomes from outpop are used for further calculations.
These three approaches (SGA, PGA with parfor loop & PGA with mapreduce) are implemented for 4 TSP instances from TSPLIB i.e. Eil51, Eil101, A280, & Oliva30. The results for all three are presented in below tables and figures.

Table 1: Optimum Tour Cost per generation for all implementations

TSP Instance	Known Optimum	SGA	PGA with *parfor*	PGA with MapReduce
Eil51	426	768	658	587
Eil101	629	1045	919	768
A280	2579	4320	3423	3145
Oliva30	423	657	611	512

Table 2: CPU Time (in msec) taken by all implementations

TSP Instance	SGA	PGA with *parfor*	PGA with MapReduce
Eil51	5182	4565	3067
Eil101	7465	6354	5342
A280	17465	11756	8624
Oliva30	4375	4165	3546

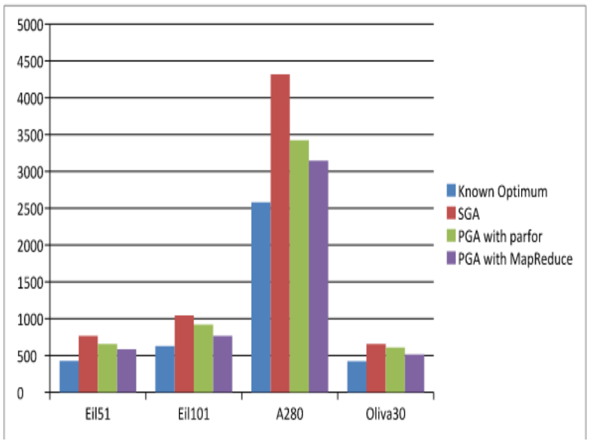

Figure 1: Optimum Tour cost found by implementations

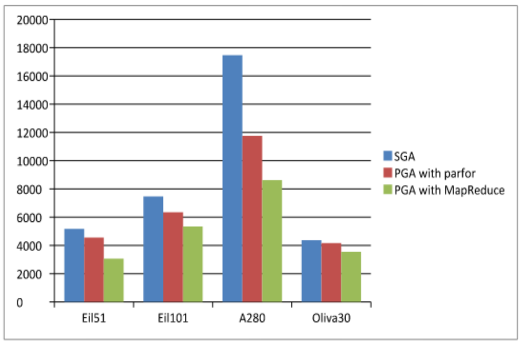

Figure 2: CPU time (in msec) for all implementations

The above results show that the implementation of GA in all aspects is able to reach close to the known optimum solutions for all instances. Further, PGA as MapReduce implementation is much better in both quality and efficiency terms in comparison to other two implementations.

5. CONCLUSIONS

GA is promising algorithm in terms of meta-heuristics and global optimization algorithms. They are used to solve many complex problems, especially from the NP-Complete category. So, it is always a problem to design better and better GA to solve a particular class of problems. If GA can be implemented in parallel then the hardware architecture can be exploited fruitfully. As the inherent nature of GA is parallel, so it is more convincing that if SGA works better in comparison to other optimization algorithms, then PGA will surely do. In this paper, PGA is implemented using two approaches with MATLAB on Intel Xeon Quad Core CPU and compared with SGA to solve TSP. It has been observed that the execution time reduces gracefully with the introduction of parallelization in GA. The optimum tour cost found by PGA is much less is comparison to SGA. Also SGA takes more time to solve the same size problem in comparison to PGA as shown in table 2. PGA implemented in this work is based on map-reduce architecture, which can be further exploited to work with more distributed and cloud based architectures. Also other approaches can be combined with PGA in its implementation like Local search or other global search.

6. REFERENCES

[1] Holland J., (1975), *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor.

[2] Rakesh Kumar, Girdhar Gopal, Rajesh Kumar, (2013), "*Hybridization in Genetic Algorithms*", International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), Vol-3, Issue-4, pp 403-409.

[3] Goldberg D. E., (1989), *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley Longman, Inc., ISBN 0-201- 15767-5.

[4] Rakesh Kumar, Girdhar Gopal, Rajesh Kumar, (2013), "*Novel Crossover Operator for Genetic Algorithm for Permutation Problems*", International Journal of Soft Computing and Engineering (IJSCE), Vol. 3, Issue-2, pp 252-258.

[5] Moscato P., Cotta C., (2003), "*A gentle introduction to memetic algorithms*", Handbook of Metaheuristics, pp 105-144.

[6] P. Jog, J. Y. Suh, and D-, Van Gucht, "*The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem*," 3rd Int'l Conference on Genetic Algorithms, july 1989, pp. 110-115

[7] Bosworth Jack, Foo Norman, and Zeigler Bernard P. (1972), "*Comparison of Genetic Algorithms with Conjugate Gradient Methods*". Technical Report 00312-1-T, University of Michigan: Ann Arbor, MI, USA.

[8] Bethke Albert Donally. (1980), "*Genetic Algorithms as Function Optimizers*". PhD thesis, University of Michigan: Ann Arbor, MI, USA.

[9] Brady R. M. (1985), "*Optimization Strategies Gleaned from Biological Evolution*." Nature, 317(6040): 804–806, doi: 10.1038/317804a0.

[10] Sinha Abhishek and Goldberg D.E. (2003), "*A Survey of Hybrid Genetic and Evolutionary Algorithms*". IlliGAL Report 2003-2004, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of Computer Science, Department of General Engineering, University of Illinois at Urbana-Champaign: Urbana-Champaign, IL, USA.

[11] Dean J., Ghemawat S., "*MapReduce: simplified data processing on large clusters*", Communications of ACM51 (2008) 107–113.