An architecture for massive parallelization of the compact genetic algorithm

Fernando G. Lobo, Cláudio F. Lima, and Hugo Mártires

ADEEC-FCT
Universidade do Algarve
Campus de Gambelas
8000-062 Faro, Portugal
{flobo,clima}@ualg.pt, hmartires@myrealbox.com

Abstract. This paper presents an architecture which is suitable for a massive parallelization of the compact genetic algorithm. The resulting scheme has three major advantages. First, it has low synchronization costs. Second, it is fault tolerant, and third, it is scalable.
The paper argues that the benefits that can be obtained with the proposed approach is potentially higher than those obtained with traditional parallel genetic algorithms.

1 Introduction

There has been several efforts in the field of evolutionary computation towards improving the genetic algorithm's efficiency. One of the efficiency enhancement techniques that has been investigated, both in theory and in practice, is the topic of parallelization [1].

With a traditional parallel genetic algorithm (GA) implementation, population members need to be sent over a computer network, and that imposes a limit on how fast they can be. In this paper, we investigate the parallelization of the compact genetic algorithm [2] [3], and take advantage of its compact representation of the population do develop a parallelization scheme which significantly reduces the communication overhead.

The paper is organized as follows. The next section presents background material on parallel GAs. Section 3 reviews the compact GA and shows the motivation for parallelizing it. Section 4 presents an architecture that allows a massive parallelization of the compact GA. In section 5 computer experiments are conducted and its results are discussed. Finally, a number of extensions are outlined, and we finish with a brief summary and the main conclusions of this work.

2 Parallel GAs

An important efficiency question that people are faced with in problem solving is the following: Given a fixed computational time, what is the best way to allocate computer resources in order to have as good a solution as possible.

Under such a challenge, the idea of parallelization stands out naturally as a way of improving the efficiency of the problem solving task. By using multiple computers in parallel, there is an opportunity for delivering better solutions in a shorter period of time.

Many computer algorithms are difficult to parallelize, but that is not the case with GAs because GAs work with a population of solutions which can be evaluated independently of one another. Moreover, in many problems most of the time is spent on evaluating solutions rather than on the internal mechanisms of the GA operators themselves. Indeed, the time spent on the GA operators is usually negligible compared to the time spent on evaluating individual solutions.

Several researchers have investigated the topic of parallel GAs and the major design issues are in choices such as using one or more populations, and in the case of using multiple populations, decide when, with who, and how often do individuals communicate with other individuals of other populations.

Although implementing parallel genetic algorithms is relatively simple, the answers to the questions raised above are not straightforward and traditionally have only been answered by means of empirical experimentation. One exception to that has been the work of Cantú-Paz [1] who has built theoretical models that lead to rational decisions for setting the different parameters involved in parallelizing GAs. There are two major ways of implementing parallel GAs:

1. Using a single population.
2. Using multiple populations.

In single population parallel GAs, also called Master-Slave parallel GAs, one computer (the master) executes the GA operations and distributes individuals to be evaluated by other computers (the slaves). After evaluating the individuals, the slaves return the results back to the master. There can be significant benefits with such a scheme because the slaves can work in parallel, independently of one another. On the other hand, there is an extra overhead in communication costs that must be paid in order to communicate individuals and fitness values back and forth.

In multiple population parallel GAs, what would be a whole population in a regular non-parallel GA, becomes several smaller populations (usually called demes), each of which is located in a different computer. Each computer executes a regular GA and occasionally, individuals may be exchanged with individuals from other populations. Multiple population parallel GAs are much harder to design because there are more degrees of freedom to explore. Specifically, four main things need to be chosen: (1) the size of each population, (2) the topology of the connection between the populations, (3) the number of individuals that are exchanged, and (4) how often do the individuals exchange.

Cantú-Paz investigated both approaches and concluded that for the case of the Master-Slave architecture, the benefits of parallelization occur mainly on problems with long function evaluation times because it needs constant communication. Multiple population parallel GAs have less communication costs but do not avoid completely the communication scalability problem. In other words,

in either approach, communication costs impose a limit on how fast parallel GAs can be. To overcome this limitation, Cantú-Paz proposed a combination of the two approaches in what was called *Hierarchical Parallel GAs*, and verified that when using such an approach it is possible to reduce the execution time more than by using either approach alone. The interested reader is referred to the original source for the mathematical formulation and for additional information on the design of parallel GAs.

The next section gives an overview of the compact GA, and after that, its parallelization is discussed.

3 The Compact Genetic Algorithm

Consider a 5-bit problem with a population of 10 individuals as shown below:

```
1 0 0 0 0
1 1 0 0 1
0 1 1 1 1
1 1 0 0 0
0 1 1 0 1
0 1 1 1 0
1 1 0 0 0
1 0 0 0 0
0 1 1 0 1
1 0 0 1 1
```

Under the compact GA, the above population can be represented by the following probability vector:

0.6	0.7	0.4	0.3	0.5

The probabilities are the relative frequency counts of the number of 1's for the different gene positions, and can be interpreted as a compact representation of the population. In other words, the individuals of the population could have been sampled from the probability vector.

Harik et al. [2] [3] noticed that it was possible to mimic the behavior of a simple GA, without storing the population explicitly. Such observation came from the fact that during the course of a regular GA run, alleles compete with each other at every gene position. At the beginning, scanning the population column-wise, we should expect to observe that roughly 50% of the alleles have value 0 and 50% of the alleles have value 1. As the search progresses, for each column, either the zeros take over the ones, or vice-versa. Harik et al. built an algorithm that explicitly simulates the random walk that takes place on the allele frequency makeup for every gene position. The resulting algorithm, the compact GA, was shown to be operationally equivalent to a simple GA that does not assume any linkage between genes.

The compact GA manipulates the population in an indirect way through the update step of $1/N$, where N denotes the population size of a regular GA. Further details about the compact GA can be found in the original source [2] [3].

3.1 Motivation for parallelizing the compact GA.

The main motivation for parallelizing the compact GA comes from the observation that the probability vector is a compact representation of the population, and it is possible to communicate the vector rather than individuals themselves. Communication costs can be reduced this way because the probability vector needs significant less storage than the whole population.

Since communication costs can be drastically reduced, it makes sense to clone the probability vector to several computers, and each computer could work independently on solving a problem by running a separate compact GA. Then, the different probability vectors would have to be consolidated (or mixed) once in a while.

This observation has first been made by Harik [4] when the compact GA was developed. The next section presents an architecture that implements Harik's idea.

4 An architecture for building a massively parallel compact GA

This section presents an architecture which is suitable for a scalable parallelization of the compact GA. Similar schemes can be done with other order-1 Probilistic Model Building Genetic Algorithms (PMBGAs) [5]. However, the connection that exists between the population size and the update step, makes the compact GA more suitable when working with very large populations, a topic that is revisited later.

The compact GA uses a probability vector as a model to represent the population. The vector can be stored with $\ell \times \log_2(N+1)$ bits (ℓ is the chromosome length, N is the population size), a different order of magnitude than the $\ell \times N$ bits needed to represent a population in a regular GA.

The storage savings are especially important when using large populations. For instance, let us suppose that we are interested in solving a 1000-bit problem using a population of size 1 million. With a regular parallel GA, in order to communicate the whole population it would be necessary to transmit approximately 1 Giga bit over a network. Instead, with the compact GA, it would only be necessary to transmit 20 thousand bits. The difference is large and suggests that running multiple compact GAs in parallel with model exchanges once in a while is something that deserves to be explored. We have devised an architecture, that we call *manager-worker*, that implements this idea. Figure 1 shows a schematic of the approach.
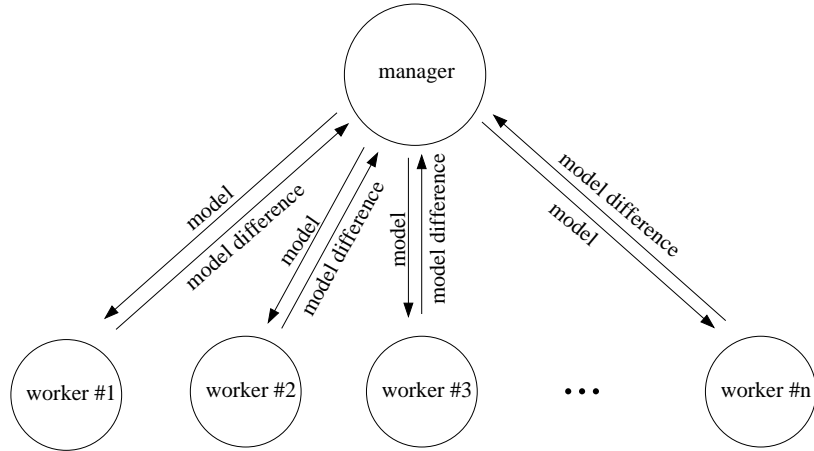
Fig. 1. Manager-worker architecture.

Although Figure 1 resembles a master-slave configuration, we decided to give
it a different name (manager-worker) to contrast with the usual master-slave
architecture of regular parallel GAs. There, the master executes and coordinates
the GA operations and the slaves just compute fitness function evaluations. In
the case of the parallel compact GA that we are suggesting, the manager also
coordinates the work of the workers, but each worker runs a compact GA on
its own. There can be an arbitrary number of workers and there is no direct
communication among them; the only communication that takes place occurs
between the manager and a worker.

4.1 Operational details

One could think of different ways of parallelizing the compact GA. The way
that we are about to propose is particularly attractive because once the manager
starts, there can be an arbitrary number of workers, each of which can start and
finish at any given point in time making the whole system fault tolerant. The
operational details consist of the following seven steps:

1. The manager initializes a probability vector of size ℓ with each variable set
   to 0.5. Then it goes to sleep, and waits to be woken up by some worker
   computer.
2. When a worker computer enters in action for the first time, it sends a signal
   to the manager saying that it is ready to start working.
3. The manager wakes up, sends a copy of its probability vector to the worker,
   and goes back to sleep.
4. Once the worker receives the probability vector, it explores m new individuals
   with a compact GA. During this period, m fitness function evaluations are

performed and the worker's local probability vector (which initially is just a copy of the manager's probability vector) is updated along the way.

5. After m fitness function evaluations have elapsed, the worker wakes up the manager in order to report the results of those m function evaluations. The results can be summarized by sending only the differences that occurred between the vector that was sent from the master and the worker's vector state after the execution of the m fitness function evaluations.

6. When the manager receives the probability vector differences sent by the worker, it updates its own probability vector by adding the differences to its current vector.

7. Then it sends the newly updated probability vector back to the worker. The manager goes back to sleep and the worker starts working for m more fitness function evaluations (back to step 4).

There are a number of subtle points that are worth mentioning. First of all, step number 7 is not a broadcast operation. The manager just sends its newly updated probability vector to one particular worker. Notice however, that the manager's probability vector not only incorporates the results of the m function evaluations performed by that particular worker, but it also incorporates the results of the evaluations conducted by the other workers. That is, while a particular worker is working, other workers might be updating the manager's probability vector. Thus, at a given point in time, workers are working with a slightly outdated probability vector. Although this might seem a disadvantage at first sight, the error that is committed by working with a slightly outdated probability vector is likely to be negligible for the overall search because an iteration of the compact GA represents only a small step in the action of the GA (this is especially true for large population sizes). The proposed parallelization scheme has several advantages, namely:

– Low synchronization costs.
– Fault tolerance.
– Scalability.

All the communication that takes place consist of short transactions. Workers do their job independently and only interrupt the manager once in a while. During the interruption period, the manager communicates with a single worker, and the other workers can continue working non-stop.

The architecture is fault tolerant because workers can go up or down at any given point in time. This makes it suitable for massively parallelization using the Internet. It is scalable because potentially there is no limit on the number of workers.

5 Computer simulations

This section presents computer simulations that were done to validate the proposed approach. In order to simplify both the implementation and the interpretation of the results, we decided to do a serial implementation of the parallel

compact GA architecture. Although it might seem strange (after all, we are describing a scheme for doing massive parallelization), doing a serial simulation of the behavior of the algorithm has a number of advantages:

– we can analyze the algorithm's behavior under careful controlled conditions.
– we can do scalability tests by simulating a parallel compact GA with a large number of computers without having the hardware.

The serial implementation that we have developed simulates that there are a number of P worker processors and 1 manager processor. The P worker processors start running at the same time and they all execute at the same speed. In addition, it is assumed that the communication cost associated with a manager-worker transaction takes a constant time which is proportional to the probability vector's size. Such a scheme can be implemented by having a collection of P regular compact GAs, each one with its own probability vector, and iterating through all of them, doing a small step of the compact GA main loop, one at a time. After a particular compact GA worker completes m fitness function evaluations, the worker-manager communication is simulated as illustrated during section 4.

We present experiments on a single problem, a bounded deceptive function consisting of the concatenation of 10 copies of a 3-bit trap function with deceptive-to-optimal ratio of 0.7 [6]. This same function has been used in the original compact GA work. We simulate a selection rate of $s = 8$ and did tests with a population size of $N = 100000$ individuals (each worker processor runs a compact GA that simulates a 100000 population size). We chose this population size because we wanted to use a size large enough to solve all the building blocks correctly. We use $s = 8$ following the recommendation given by Harik et al. in the original compact GA paper for this type of problem. Finally, we chose this problem as a test function because, even though the compact GA is a poor algorithm in solving the problem, we wanted to use a function that requires a large population size because those are the situations where the benefits from parallelization are more pronounced.

Having fixed both the population size and the selection rate, we decided to systematically vary the number of worker processors P, as well as the m parameter which has an effect on the rate of communication that occurs between the manager and a worker. We did experiments for P in $\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$, and for a particular P, we varied the parameter m in $\{8, 80, 800, 8000, 80000\}$. This totalled 55 different configurations, each of which was run 30 independent times.

The m parameter is important because it is the one that affects communication costs. Smaller m values imply an increase in communication costs. On the other hand, for very large m values, performance degrades because the compact GA workers start sampling individuals from outdated probability vectors.

Figure 2 shows the results. In terms of fitness function evaluations per processor, we observe a linear speedup for low m values. For instance, for $m = 8$ we observe a straight line on the log-log plot. Using the data directly, we calculated the slope of the line and obtained an approximate value of -0.3. In order
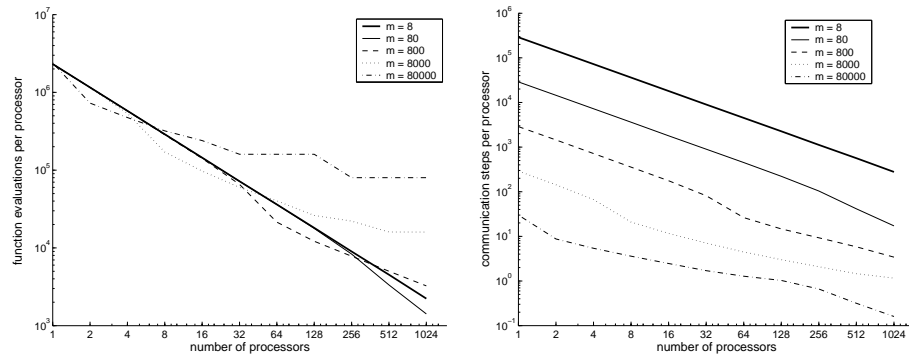
Fig. 2. Both graphs depict a log-log plot. On the left, we see the average number of function evaluations per processor. On the right, we see the average number of communication steps per processor.

to take into account the different logarithm bases, we need to multiply it by $\log_2 10$ (y-axis is $\log_{10}$, x-axis is $\log_2$) yielding a slope of approximately -1. This means that the number of function evaluation per processor decreases linearly with a growing number of processors. That is, whenever we double the number of processors, the average number of fitness function evaluations per processor gets cut by a half.

Likewise, in terms of communication costs, as we raise the parameter m, the average number of communication steps between manager and worker decreases in the same proportion as expected. For instance, for $m = 80$, communication costs are reduced 10 times when compared with $m = 8$. Notice that there is a degradation in terms of speedup for the larger m values. For instance, for $m = 8000$ and $m = 80000$ (which is about the same order of the population size), the speedup obtained goes away from the idealized case. This can be explained by the fact that in this case (and especially with a large number of processors), the average number of communication steps per processor approaches zero. That means that a large fraction of processors were actually doing some work but never communicated their results back to the manager because the problem was solved before they had a chance to do so.

6   Extensions

This work has a number of extensions worthwhile exploring. Below, we outline some of them:

- Build theory for analyzing the effect of m, N, and P.
- Compare with traditional parallel GA schemes.
- Extend the approach to multivariate PMBGAs.
- Take advantage of the Internet and build something like SETI@home.

It would be interesting to study the mathematical analysis of the proposed parallel compact GA. A number of questions come to mind. For instance, what is the effect of the m parameter? What about the number of workers P? Should m be adjusted automatically as a function of P and N? Our experiments suggest that there is an "optimal" m that depends on the number of compact GA workers P, and most likely depends on the population size N as well.

Another extension that could be done is to compare the proposed parallel architecture with those used more often in traditional parallel GAs, either master-slave and multiple deme GAs. Again, our experiments suggest that the parallel compact GA is likely to be on top of regular parallel GAs due to lower communication costs.

The model structure of the compact GA never changes, every gene is always treated independently. There are other PMBGAs that are able to learn a more complex structure dynamically as the search progresses. One could think of using some of the ideas presented here for parallelizing these more complex PMBGAs.

Finally, it would be interesting to have a parallel compact GA implementation based on the Internet infrastructure, where computers around the world could contribute with some processing power when they are idle. Similar schemes have been done with other projects, one of the most well known is the SETI@home project [7]. Our parallel GA architecture is suitable for a similar kind of project because computers can go up or down at any given point in time.

7   Summary and conclusions

This paper reviewed the compact GA and presented an architecture that allows its massive parallelization. The motivation for doing so has been discussed and a serial implementation of the parallel architecture was simulated. Computer experiments were done under idealized conditions and we have verified an almost linear speedup with a growing number of processors.

The paper presented a novel way of parallelizing GAs. This was possible due to the different operational mechanisms of the compact GA when compared with a more traditional GA. By taking advantage of the compact representation of the population, it becomes possible do distribute its representation to different computers without the associated cost of sending it individual by individual.

Additional empirical and theoretical research needs to be done to confirm our preliminary results. Nonetheless, the speedups observed in our experiments suggest that a massive parallelization of the compact GA may constitute an efficient and practical alternative for solving a variety of problems.

References

1. Cantú-Paz, E.: Efficient and accurate parallel genetic algorithms. Kluwer Academic Publishers, Boston, MA (2000)

2. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. In: Proceedings of 1998 IEEE International Conference on Evolutionary Computation, Piscataway, NJ, IEEE (1998) 523–528
3. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. IEEE Transactions on Evolutionary Computation **3** (1999) 287–297
4. Harik, G.R.: Personal communication. (1997)
5. Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications **21** (2002) 5–20 Also IlliGAL Report No. 99018.
6. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. In Whitley, L.D., ed.: Foundations of Genetic Algorithms 2, San Mateo, CA, Morgan Kaufmann (1993) 93–108
7. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Lebofsky, M.: SETI@home - massively distributed computing for SETI. Computing in Science and Engineering **3** (2001) 79