# A Summary of Research on Parallel Genetic Algorithms

**Erick Cantú-Paz**

IlliGAL Report No. 95007
July 1995

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-0897
Fax: (217) 244-5705

# A Summary of Research on Parallel Genetic Algorithms

**Erick Cantú-Paz**
Computer Science Department and
The Illinois Genetic Algorithms Laboratory (IlliGAL)
University of Illinois at Urbana-Champaign
cantupaz@uiuc.edu

**Abstract**

The main goal of this paper is to summarize the previous research on parallel genetic algorithms. We present an extension to previous categorizations of the parallelization techniques used in this field. We will use this categorization to guide us through a review of many of the most important publications. We will build on this survey to try to identify some of the problems that have not been studied systematically yet.

## 1 Introduction

Genetic Algorithms (GAs) are efficient search methods based on principles of natural selection and population genetics. They are being successfully applied to problems in business, engineering and science (Goldberg, 1994). GAs use randomized operators operating over a population of candidate solutions to generate new points in the search space.

In the past few years, parallel genetic algorithms (PGAs) have been used to solve difficult problems. Hard problems need a bigger population and this translates directly into higher computational costs. The basic motivation behind many early studies of PGAs was to reduce the processing time needed to reach an acceptable solution. This was accomplished implementing GAs on different parallel architectures. In addition, it was noted that in some cases the PGAs found better solutions than comparably sized serial GAs.

Today, the study of parallel GAs is flourishing. GAs are easy to parallelize and many variants on the basic models have been tried with good results on different classes of problems. However, most of the research has been empirical and, until recently, we lacked a theory that helped resolve the fundamental questions that arise in the design of these algorithms.

The goal of this report is to review the previous efforts in the parallelization of GAs and try to expose some of the problems that still have not been solved satisfactory. To guide us in this review, we will present a categorization of the parallelization techniques used over the years.

The next section is a brief introduction to genetic algorithms. Section 3 describes our categorization of parallel GAs. The bulk of this report is the sections that review the most relevant papers in the literature. These sections are followed by a description of some open research problems in the field. We end this report with a summary and the conclusions of this project.

## 2 Genetic Algorithms – A Quick Introduction

Simple GAs operate on a fixed-sized population of fixed-length individuals. The individuals are a binary string that encodes the variables of the problem that the algorithm is trying to optimize.

Simple GAs use three operators: selection, crossover and mutation.

The **selection** operator identifies the fittest individuals of the current population to serve as parents of the next generation. The fitness value of each individual is given by a problem-dependent function. The selection mechanism can take many forms, but it always ensures that the best individuals have a higher probability to be selected to reproduce to form a new generation.

The primary exploration mechanism for GAs is **crossover**. This operator randomly chooses a pair of individuals among those previously selected to breed and exchanges substrings between them. The exchange occurs around randomly selected crossing points.

The **mutation** operator is usually considered a secondary operator. Its main function is to restore diversity that may be lost from the repeated application of selection and crossover. This operator simply takes one string from the population and randomly alters some value within it. Following nature's example, the probability of applying the mutation operator is very low compared to the probability of applying the crossover operator.

It has been established that the time required by a GA to converge is $O(n \log n)$ function evaluations (Goldberg, 1991), where $n$ is the population size. We say that a population has converged when all the individuals are very much alike and further improvement may only be possible by a favorable mutation.

Genetic algorithms are not *guaranteed* to find an *optimal* solution and their effectivity is determined largely by the population size $n$ (Goldberg, 1991). As the population size increases, the GA has a better chance of finding the global solution, but as we saw above, the computation cost also increases as a function of the population size.

With serial GAs, we have to choose between getting a good result with a high confidence and pay a high computational cost or loosen the confidence requirement and get (possibly poor) results fast. In contrast, parallel GAs can keep the quality of the results high and find them fast because, using parallel machines, larger populations can be processed in less time. This keeps the confidence factor high and the response time low, opening opportunities to apply genetic algorithms in time-constrained applications. Additionally, parallel GAs may evolve several different independent solutions that may be recombined at later stages to form better solutions.

## 3 A Classification of Parallel GAs

Evolution is a highly parallel process. Each individual is selected according to its fitness to survive and reproduce. GAs are an abstraction of the evolutionary process and are indeed very easy to parallelize.

As a first step in our investigation of parallel GAs, we must recognize that there are different ways to parallelize GAs. This section presents a classification of the efforts made up to date. Similar classifications appear in (Adamidis, 1994; Gordon and Whitley, 1993; Lin, Punch, and Goodman, 1994), however in this paper we extend them to include one more category. We will use this classification to guide our review of parallel GAs publications in the following sections.

The first approach in parallelizing GAs is to do a **global** parallelization. In this class of parallel GAs, the evaluation of individuals and the application of genetic operators are explicitly parallelized. Every individual has a chance to mate with all the rest (i.e., there is random mating). Therefore, the semantics of the operators remain unchanged. This method is relatively easy to implement and a speedup proportional to the number of processors can be expected. In section 5 we will describe strategies for implementing this model in shared and distributed memory machines.

A more sophisticated idea is used in **coarse grained**[1] parallel GAs. The population is divided

[1] The "grain size" in parallelism refers to the ratio of the time spent in computation and the time spent in communication. When this ratio is high the processing is called coarse grained.

into a few subpopulations keeping them relatively isolated from each other. This model of parallelization introduces a **migration** operator that is used to send some individuals from one subpopulation to another.

Two population genetics models for population structures are used in different implementations of coarse grained GAs: the island model and the stepping stone model. The population in the island model is partitioned into small subpopulations by geographic isolation and individuals can migrate to *any* other subpopulation. In the stepping stone model, the population is partitioned in the same way, but migration is restricted to neighboring subpopulations. Both models have been used in parallel GAs and we will not make any distinction between them in our review. Sometimes coarse grained parallel GAs are known as "distributed" GAs since they are usually implemented in distributed memory MIMD computers.

The third approach in parallelizing GAs uses fine grained parallelism. **Fine grained** parallel GAs partition the population into a large number of very small subpopulations. The ideal case is to have just one individual for every processing element available. This model calls for massively parallel computers.

In the last two approaches, selection and mating occur only within each subpopulation. Based on this we will borrow from Biology the term **deme** to refer to a subpopulation. Since the size of the demes is usually smaller than the population used by a serial GA, we can expect that the PGA will converge faster. However, when we consider that there is communication between the demes it is no longer clear whether the PGA will converge faster or not.

It is important to notice that while the global model did not affect the behavior of the algorithm, the last two approaches are introducing fundamental changes in the way the GA works. In subsequent sections we will identify these changes and we will begin an investigation that should lead to better ways to exploit parallelism in GAs.

The final method to parallelize GAs uses some combination of the first three methods. We will call this class of algorithms **hybrid** parallel GAs. There are not many papers published describing implementations with hybrid techniques, but some of them have achieved very interesting results as we will see in section 8.

# 4 Early Ideas

In the late fifties, John Holland (1959,1960) proposed a class of parallel machines that would be able to run an undetermined number of programs concurrently. These machines were never built, but Holland recognized the parallel nature of the evolutionary process and mapped it to the machine he designed.

Probably the first attempt to map genetic algorithms to existing parallel computer architectures was made in 1981 by John Grefenstette (Grefenstette 1981). He proposed four implementations of parallel GAs. The first implementation (which Grefenstette called synchronous master-slave) is like the global parallelization model described above. The second implementation is similar to the first, but the evaluation of the individuals is done asynchronously (i.e. there is no clear division between generations, when any processor finishes evaluating an individual it returns it to the population and receives another individual). The third prototype keeps the population in a shared memory that can be accessed by some processors that do function evaluations and apply genetic operators independently of each other. This is also a global PGA in our classification. The fourth prototype is a coarse grained PGA, where the best individuals are broadcast every generation to all the other processors. This is the only prototype that was implemented some time later (Pettey, Leuze, and Grefenstette, 1987) and we will review some of its results in the next section.

## 5 Global parallelization

Recall that in this model there is a single population and the evaluation of the individuals and sometimes the application of the genetic operators is done in parallel.

The evaluation can be parallelized assigning a subset of individuals to each of the processors available. There is no communication between the processors during the evaluation because the fitness of each individual is independent from all the others. Communication only occurs at the start and at the end of the evaluation phase.

On a shared memory multiprocessor, the individuals could be stored in shared memory. Each processor can read the individuals assigned to it and write the evaluation results back without any conflicts[2]. Note that there is some synchronization needed between generations. In some cases (like in a multiuser environment) it may be necessary to balance the computational load among the processors using a dynamic scheduling algorithm (e.g., guided self-scheduling). In any case, a speedup proportional (but probably sublinear) to the number of processors can be expected.

On a distributed memory computer, the population can be stored in one processor to simplify the application of the genetic operators. This "master" processor would be responsible for sending the individuals to the other processors (the "slaves") for evaluation, collecting the results, and applying the genetic operators to produce the next generation. There would be a bottleneck in the algorithm since the slave processors sit idle while the master is doing its work.

An example of global parallelization is the work of Fogarty and Huang (1990). They used a transputer network to evolve a set of rules for a pole balancing application. The simulation of the cart and the pole takes a considerable time. They connected the transputers in different topologies, but they conclude that in cases where the computation time dominates over the communication time, the configuration of the network is not important. They noted that the speedups were sublinear because the communication overhead increased very fast as they added more transputers to the network.

Another more recent implementation of a Global GA is the work by Hauser and Männer (1994). They used three diferent parallel computer architectures, but only got reasonable speedups on a NERV multiprocessor (speedup of 5 using 6 processors), that has a very low overhead. On the other systems they used (a SparcServer and a KSR1), they did not get good speedups because the thread libraries they used did not gave them complete control over the scheduling of threads to processors.

A further step in global parallelization is to apply the genetic operators in parallel. We will examine each of the operators of the simple GA. First, consider the selection operator. There are several variants of selection and some require a global statistic (like the population average) which could introduce a serial bottleneck in the algorithm. Fortunately, if we use some form of tournament selection we only need the fitness of a subset of the individuals (usually only two). Now consider crossover, it can be applied simultaneously over $n/2$ pairs of individuals. Mutation can be applied to each individual (and even to each bit) independently of the others.

On the other hand, it is not clear whether parallelizing the application of the operators would result on a performance improvement or not. The genetic operators are very simple and the time spent in communication could easily be longer than the time doing any computation. This is specially true in distributed memory machines, where the overhead for each message might be considerable.

Abramson (1992) implemented a GA on a shared memory computer (an Encore Multimax with 16 processors) to search for efficient timetables for schools. He parallelized the creation and evaluation of individuals. Abramson reported sublinear speedups, and blamed a few sections of serial code on the critical path of the program for the results. He later (1993) used a distributed memory machine

[2]We mean that there are no conflicts with other processors to access the same memory locations and thus no synchronization is required in this step, but there may be conflicts in the interconnection network that may slow the algorithm.

(a Fujitsu AP1000 with 128 processors) and changed his application to train timetables. In both systems, he reports almost linear speedups up to 16 processors. In the distributed memory machine the speedup degrades fast with more processors probably for the reasons outlined above.

# 6 Coarse Grained PGAs

The important characteristics of this class of algorithms are the use of few relatively large demes and the introduction of a migration operator. Coarse grained parallel GAs are the most popular model and many papers have been written describing many aspects and details of their implementation. We will give a thorough review of this area, but we will concentrate on the most influential papers.

## 6.1 The First Generation

One of the pioneering efforts is Grosso's dissertation (1985). He simulated diploid individuals and the population was divided in five demes. The demes exchanged individuals using a fixed migration rate and several rates were tried in the experiments. Grosso found that the rate of improvement was faster in the smaller demes than in a single large panmictic population, confirming the views of Sewall Wright. However, when the demes were isolated, this rapid rise in fitness stopped at a lower level than with the larger population. At intermediate migration rates, the divided population displayed a behavior similar to the panmictic population. With a lower migration rate, the demes had the opportunity to behave independently and explore different regions of the search space. But, if the migration rate is to low, the migrants might not have a significant effect on the receiving deme. These results point out that there is a critical migration rate below which the performance of the algorithm is obstructed by the isolation of the demes and above which the partitioned population behaves as a panmictic one.

In the first International Conference of Genetic Algorithms (ICGA), there were no papers about parallel GAs at all. This situation changed in the second ICGA in 1987, where six papers were published. From then on there has been a steady flow of papers published in conferences and journals of GAs and parallel computation. Very influential papers were published in 1987 and, as we will see, some of the ideas and problems presented in those papers are still an open area of research.

Tanese (1987) proposed a parallel GA that used a 4-D hypercube topology to communicate individuals from one deme to another. In Tanese's algorithm migrations occurred at uniform periods of time between neighbor processors along one dimension of the hypercube. The migrants were chosen probabilistically from the best individuals in the subpopulation and they replaced the worst individuals in the receiving deme. Tanese reported that the parallel GA found results as good as a serial GA, with the advantage of near-linear speedups.

Also in this year, Cohoon, Hedge, Martin, and Richards (1987) proposed an implementation of a parallel GA based on the theory of "punctuated equilibria". This theory was proposed by Elredge and Gould to explain the (apparently) missing links in the fossil record. One aspect of the punctuated equilibria theory is that new species are likely to form quickly in relatively small isolated populations after some change in the environment occurs. The genetic composition of a population is an important part of an organism environment and thus, immigration can be a trigger for evolutionary changes. Cohoon et al. noticed that the number of migrants affected the level of disruption in the demes and that new solutions were found shortly after migration occured.

A linear placement problem was used as a benchmark and experimented using a mesh topology. However, they noted that the choice of topology is probably not very important in the performance of the parallel GA as long as it has "high connectivity and a small diameter to ensure adequate

'mixing' as time progresses". Note that the mesh topology they chose has arity four and the diameter is $O(\sqrt{n})$.

Keeping the total cost constant, they found that the parallel GA with migration outperformed both a parallel GA without migration and a serial GA. This work was later extended by the same group using a VLSI application (a graph partitioning problem) on a 4-D hypercube topology (Cohoon, Martin, and Richards, 1991a, 1991b). Note that the nodes in the hypercube also have four neighbors and the diameter is $\log_2 n$ (shorter than the mesh).

The last paper we will examine from 1987 uses parallelism to deal with the problem of getting fast results while using increasingly larger population sizes that are needed in some applications (Pettey, Leuze, and Grefenstette, 1987). This is a direct implementation of Grefenstette's fourth prototype outlined in the previous section. In this algorithm, a copy of the best individual found in each deme is sent to all its neighbors after every generation to ensure good mixing. In this paper, Pettey, Leuze, and Grefenstette recognize that this approach can be regarded as equivalent to a sequential GA with a single large panmictic population, with the only difference being that selection occurs at the deme level.

Tanese (1989a) continued her work making a very exhaustive experimental study on the frequency of migrations and the number of individuals exchanged. She found that migrating too many individuals too frequently or too few individuals very infrequently degraded the performance of the algorithm. However, good quality results were found faster than with a serial GA even in cases with no migration at all. More details about the experiments and the conclusions she derived can be found in her dissertation (Tanese, 1989b).

A recent paper by Belding (Belding, 1995) attempts to extend Tanese's work using the Royal Road functions. Migrants were sent to a random destination, rather than using a hypercube topology, as in Tanese's original study, but the author claims that experiments with a hypercube yielded similar results. In most cases, the global optimum was found more often when migration was used than in the completely isolated cases. Surprisingly, convergence was faster when a high migration rate was used ($r = 0.5$) than with a low rate.

### 6.2 The Next Generation

From the papers examined above, we can recognize some very important issues emerging. First, migration is controlled by several parameters; the **topology** that defines the connections between the subpopulations, a **migration rate** that controls how many individuals migrate, and a **migration interval** that affects how often migrations occur. Second, the values for these parameters are chosen using intuition rather than analysis. Although some tried to make some analysis on PGAs, intuition continued to be the norm in the following years.

Probably the first attempt to provide some theoretical foundations to the performance of a parallel GA is a 1989 paper by Pettey and Leuze (1989). In this article they describe a derivation of the schema theorem for parallel GAs. They show that the expected number of trials allocated to schemata can be bounded by an exponential function when randomly selected individuals are broadcast every generation.

It can be expected that relatively isolated demes will find different partial solutions to a given problem. Starkweather, Whitley, and Mathias (1991) made an important observation regarding this: if partial solutions can be combined to form a better solution, then a parallel GA would probably outperform a serial GA. On the other hand, if the recombination of partial solutions results in a less fit individual, then the serial GA might have an advantage.

One of the papers that showed the potential advantages of parallel GAs was Mühlenbein's 1991 ICGA paper. It describes a multi-population GA applied to difficult function optimization problems. The PGA was able to find the global optimum of very large problems. Later on, the functions used

in this paper were adopted by other researchers for their empirical work (e.g. Chen, Meyer, and Yeckel, 1993; Gordon and Whitley, 1993). Unfortunately, Mühlenbein included a local optimizer in his algorithm, and although he managed to design an efficient and powerful algorithm, it is not clear whether the results obtained can be credited to the distributed population or to the local optimizer.

In his publications, Mühlenbein (1989, 1991a, 1991b, 1991d) has made it clear that he believes that PGAs can be tools in the study of evolution of natural organisms. This view is shared by others, but the inclusion of a local optimizer makes it impossible to study the effect of the genetic operators on the genetic composition of the populations.

Note that in all the papers cited so far, migration occurs at predetermined constant intervals. A different approach was introduced in a 1990 paper presented at the first Parallel Problem Solving from Nature workshop (PPSN) where migration occurs after the subpopulations converge (the author uses the term "degenerate") (Braun, 1990). The same concept was later used (with some alterations) by Munetomo, Takai, and Sato (1993). These researchers raised an important (and yet unanswered) question: when is the right time to migrate? If migration occurs too early the number of correct building blocks in the migrants may be too low to influence the search on the right direction and we would be wasting expensive communication resources.

As we have seen, the bulk of the work on parallel GAs has been empirical, but few researchers have identified and studied a particular issue. A traditionally neglected aspect of parallel GAs has been the topology of the interconnection between demes. The topology is an important factor in the performance of the parallel GA, because it determines how fast (or how slow) a good solution disseminates to other demes. If the topology has a dense connectivity (or a short diameter, or both), good solutions will spread fast to all the demes and may quickly take over the population. On the other hand, if the topology is sparsely connected (or has a long diameter), solutions will spread slower and the demes will be more isolated from each other, permitting the appearance of different solutions. These solutions may come together at a later time and recombine to form potentially better individuals.

Most implementations of coarse grained parallel GAs use the native topology of the computer available to the researchers. For example, implementations on hypercubes (Cohoon, Martin, and Richards, 1991; Tanese, 1989; Tanese1989) and rings (Gordon and Whitley, 1993) are common.

Bianchini and Brown (1992, 1993) attempted to tackle the topology problem considering the problems and restrictions that arise in a transputer implementation These two researchers tried different topologies and presented experimental results that support the idea of using isolated and semi-clustered demes. These topologies have low connectivity (making them realizable with transputers which can connect to only four other transputers) and good results were found using them. A more recent empirical study of different topologies showed that topologies with higher arity find the global solution faster than the more sparsely connected ones (Cantú-Paz and Mejía- Olvera 1994). This study considered completely connected topologies, 4-D hypercubes, a $4 \times 4$ toroidal mesh and unidirectional and bidirectional rings.

An unconventional approach to migration was recently developed by Marin, Trelles-Salazar, and Sandoval (1994). They propose a centralized model where the processes that are executing the GAs periodically send their best partial results to a master process. Then, the master process chooses the fittest individuals among those that it receives and broadcast them to all the nodes. Experimental results show that near linear speedups can be obtained with a small number of nodes (around six), and the authors claim that this approach can scale up to larger workstation networks.

### 6.3 Non-traditional GAs

Some non-traditional GAs have also been parallelized and some improvement has been reported over their serial versions. Maresky (1994) used Davidor's ECO model as a basis of his distributed

algorithm. Each node in this algorithm is a ECO GA. Maresky explored different migration schemes with varying complexity. The simplest scheme is to do no migration at all and just collect the results from each node at the end of a run. The more complex algorithms involve the migration of complete ECO demes. Several replacement policies were explored with only marginal benefits.

GENITOR is a "steady-state" GA where only a few individuals change in every generation. In the distributed version of GENITOR (Starkweather, Whitley, and Mathias, 1991), individuals migrate at fixed intervals to neighboring nodes. Immigrants replace the worst individuals in the target deme. A significant improvement over the serial version was reported in the original paper and in (Gordon and Whitley, 1993).

There have also been efforts to parallelize **messy GAs**. Messy GAs have two phases. In the *primordial* phase the initial population is created using a partial enumeration and it is reduced using tournament selection. Next, in the *juxtapositional* phase partial solutions found in the primordial phase are mixed together. The primordial phase dominates execution time and several data distribution strategies were tried by Merkle and Lamont (1993) extending the work reported previously by Dymek (1992). The results indicate that the quality of the solutions was not statistically different in most of the cases varying the distribution strategy, but that gains in execution time could be possible. The parallel fast messy GA was used to search for optimal conformations of Met-Enkephalin molecules (Merkle, Gates, Lamont and Pachter, 1993). In this problem, the algorithm showed sublinear speedups with up to 32 processors, after that the execution time started to increase.

More recently, Koza and Andre (1995) used a network of transputers to parallelize a Genetic Programming application. In their report they make an analysis of the requirements of their algorithm and choose transputers as the most cost-effective solution to implement it. The report also includes a very detailed description of the implementation of their algorithm. They used an island model and experimented with different migration rates using 64 demes. They reported super linear speedups over the panmictic version using moderate migration rates (around 8 percent).

### 6.4 Applications

Many application projects using coarse grained parallel GAs have been published. The graph partitioning problem has been attacked by Béssiere (1991), Cohoon et al. (1991) and Talbi and Béssiere (1991a, 1991b). Levine's (1994) work on the graph partitioning problem is outstanding. His algorithm found global solutions of problems with 36,699 and 43,749 integer variables. He found that performance, measured as the the quality of the solution and the iteration on which it was found, increased as more demes were added. The quadratic assignment problem, another combinatorial optimization application, has been solved by Li and Mashford (1990) and Mühlenbein (1989).

The problem of load distribution of processes to processors in MIMD architectures has been solved successfully by Neuhaus (1990) and by Muntean and Talbi (1991).

## 7 Fine Grained PGAs

In this section we will examine some publications on the fine grained approach to parallelize GAs. Recall that in this model the population is divided into many small demes. The demes overlap providing a way to disemminate good solutions across the entire population. Again, selection and mating occur only within a deme.

In the second ICGA, Robertson (1987) published a paper describing the parallelization of a GA in a classifier system using a Connection Machine. He parallelized the selection of parents and of classifiers to replace and also the mating and crossover operations. The execution time of this algorithm is independent of the number of classifiers (up to 16K).

In 1989 the ASPARAGOS system was introduced in two papers by Gorges-Schleuter (1989) and Mühlenbein (1989). It uses a population structure that looks like a ladder with the upper and lower ends tied together (forming a ring). ASPARAGOS was used to solve some difficult combinatorial optimization problems with great success. Later, a linear variation on the population structure was tried by Gorges-Schleuter (1990). There is no question on the effectiveness of this system, but because it uses non-conventional genetic operators (like the PMX crossover) and a hill-climbing algorithm to further optimize the individuals in the population, it is difficult to use the results from this system to help us understand the dynamics of a fine grained PGA. A comparison of different mating strategies was discussed in a later paper (Gorges-Schleuter, 1992).

Another massively parallel GA with a more traditional population structure was introduced by Manderick and Spiessens (1989) and studied further in subsequent years (Spiessens and Manderick, 1990, 1991). In this algorithm the population is distributed on a 2-D mesh topology. Selection and mating are only possible with neighboring individuals, therefore the demes are defined by the spatial distribution of the individuals. The authors noted that the performance of the algorithm degraded as the size of the deme increased, On the extreme, if the size of the neighborhood was big enough, this parallel GA was reduced to a conventional panmictic population. Their analysis showed that, for a deme size $s$ and a string length $l$, the time complexity of the algorithm was $O(s+l)$ or $O(s(\log s)+l)$ time steps depending on the selection scheme used.

Davidor (1991) also used 2-D grid to place the individuals, but on his algorithm offspring are placed in their parents' deme. A conflict occured if the grid element is occupied by another individual and it is solved probabilistically using the fitness of both individuals as bias. Davidor named his algorithm the "ECO model".

It is common to place the individuals of a fine grained PGA in a 2-D grid because in many massively parallel computers the processing elements are connected using this topology. However, most of these computers have a global router that can send messages to any processor in the network (at a higher cost) and other topologies can be simulated on top of the grid. There is a study that compares the effect of using different interconnection networks on fine grained GAs (Schwehm, 1992). In this paper a graph partitioning problem is used as a benchmark to test different topologies simulated with a MasPar MP-1 computer. The topologies tested were a ring, a torus, a $16 \times 8 \times 8$ cube a $4 \times 4 \times 4 \times 4 \times 4$ cube, and a 10-D binary hypercube. The results for this problem showed that a torus had a better performance than the other higher or lower dimensionality topologies.

Anderson and Ferris (1990) tried different topologies and different replacement algorithms. They experimented with two rings, a hypercube, two meshes and an "island" topology where only one individual of each deme overlapped with other demes. They concluded that for the particular problem they were trying to solve (assembly line balancing) the ring and the "island" topologies were the best.

Baluja (1992, 1993) also compared different topologies with different diameters. He tried two variations on a linear topology and a 2-D mesh. The mesh had the shortest diameter and gave the best results on almost all the problems tested.

From the results of these papers we can conclude that the topology affects the performance of the algorithm. It seems that a topology with a medium diameter gives good results.

Some application problems that use fine grained parallel GAs have been published. A popular problem is two dimensional bin packing and it has been solved successfully by Kroger (1990, 1992a, 1992b, 1993). The job shop scheduling problem is very popular in the GA literature and Tamaki and Nishikawa (1992) solved it using a fine grained parallel GA.

Shapiro and Navetta (1994) succesfully applied a fine grained algorithm to predict the secondary structure of RNA. They used the native X-Net topology of the MasPar-2 computer to define the nieghborhood of each individual. The processing elements on the MasPar-2 are placed on a 2-D

grid and are directly connected to their eight closest neighbors. Diferent logical topologies were tested with the GA: all eight nearest neighbors, four nearest neighbors, and all neighbors within a specified distance $r$. The results for $r > 2$ were the poorest and the four neighbor scheme consistently outperformed the eight neighbors topology.

A few papers compare the performance of fine and coarse grained parallel GAs. The conclusions of these papers are inconclusive, mainly because, as Baluja (1993) states in his paper, it is very difficult to come up with a measure of performance that can be used to compare the two classes of algorithms directly. Baluja's comparison of three fine grained algorithms with a coarse grained favoured the former, but in Gordon and Whitley's paper (1993), the results were more inclined towards the coarse grained algorithms.

In an earlier paper, Gordon and Whitley (1992) showed that the critical path of a fine grained algorithm is shorter than that of a coarse grained GA. This means that if enough processors are available, massively parallel GAs will need less time steps to finish regardless of the population size. However, it is important to note that this was a theoretical study that did not include considerations such as the communication bandwidth between processors or memory requirements.

# 8 Hybrid Algorithms

A few researchers have tried to combine two of the methods to parallelize GAs. Some of these new hybrid algorithms add a new degree of complexity to the already complicated scene of parallel GAs, but other algorithms manage to keep the same complexity as one of their components.

Gruau (1994) invented a "mixed" parallel GA. In his algorithm the population of each deme is distributed in a 2-D grid. The demes themselves are connected as a 2-D torus and migration between neighboring demes occurs at regular intervals. Good results were reported for a novel neural network design and training application.

Another way to hybridize a parallel GA is to use global parallelization on each of the demes of a coarse grained GA. Migration occurs between demes as in the coarse grained algorithm, but the evaluation of the individuals is handled in parallel. This approach does not introduce new analytic problems and can be useful when working with complex applications with objective functions that need a considerable amount of computation time.

This approach has not been used yet in any application, but it does not introduce more complexity into its analysis. Recall that the global parallelization model has the same properties as a serial GA, thus the analysis for this hybrid model will follow the exact same path as the analysis for coarse grained PGAs. Good speedups can be expected from this method when applied to problems that need a long time to be evaluated and extremely large populations.

# 9 Underlying Problems

We have seen in previous sections that parallel GAs are effective in solving a number of difficult problems and that they can be mapped to different kinds of parallel machines. Parallel GAs promise a lot both on terms of speedups and in improving the search quality of the serial GAs. With all the good results published it is easy to miss that there are several issues that are not understood at all.

This section will summarize the most important problems and the advances in their understanding. We will focus on coarse grained GAs, since they are the more realistic simulation of natural processes, they are better understood than fine grained GAs, and they are the most popular approach.

To start our review of the problem we have to acknowledge that it has several facets. A first step in a formal investigation of parallel GAs is to decompose the problem into manageable parts

that can be studied independently and that we can reunite later. A simple first level decomposition divides the study of parallel GAs in two parts: population sizing and migration.

The size of the population is probably the parameter that affects most the performance of the GA. There is a population sizing theory for serial GAs that is based on statistical decision making theory and gives conservative lower bounds on the population sizes needed to solve problems with a given confidence factor (Goldberg, Deb, and Clark, 1992) In a parallel GA, the population sizing question is very closely related to the *deming* issue. That is, how many demes and of what size does the parallel GA need to solve a problem with a given confidence? This questions have been addressed recently (Goldberg, Kargupta, Horn, & Cantú-Paz, 1995) and solved for the extreme cases where perfect mixing between the demes is assumed and for isolated demes.

After we determine the number and the size of demes we will use, we have to establish how they are going to communicate. Migration should be the next step in the investigation of parallel GAs. It is a very complex operator and as we saw in section 6, the migration of individuals across demes is affected by several parameters. To be able to study migration, we have to study each of its facets independently, but in a way that allows for the reconstruction of the whole issue again. In other words, we need a second decomposition.

A possible decomposition of the migration parameter could be:

- A migration timescale. Intuitively, migration should occur after the expected number of building blocks in each individual is relatively high. Migrating before that is probably a waste of (communication) resources.
- Migration rate. The number of individuals that we should migrate is very closely related with the migration timescale. If individuals are 'rich' in building blocks, maybe we can get good results migrating just a few.
- Topology. What is the best way to connect the demes? If we use a topology with a long diameter, good solutions will take a longer time to propagate to all the demes. On the other hand, if we use a topology that retards the mixing of partial solutions, different 'species' might appear at they may not mix to produce a better solution.

These three points are still open areas of research. We have reviewed several papers that favor some migration intervals, migration rates, or topologies over anothers, but there is still no conclusive analysis of the impact of these parameters on the search quality or in the speed of the algorithm.

## 10 Summary and Conclusions

In this paper we reviewed the existing literature on parallel genetic algorithms. We started by classifying the work in this field into four categories: global parallelization, coarse and fine grained algorithms, and hybrid approaches.

We analysed the most important contributions in each of these categories, trying to identify the issues that affect the design of these algorithms and their implementation on currently available parallel computers. We found that the research in this field is dominated by the description of experimental results and that very little work has been conducted to give an analytical explanation of what is observed.

We focused on coarse grained parallel GAs, partly because of their popularity, but mainly because of their close resemblance to nature. For this class of algorithms, the spatial allocation of resources (i.e., the decision on the number and the size of demes) has been solved for extreme cases, but the migration issue is still unresolved.

Parallelization of GAs is not a very difficult task and indeed many implementations are reported in the literature. Maybe too many implementations. What we need now are studies that focus on particular aspects of parallel GAs. We need to understand what truly affects the performance of the algorithm so we can design faster and more reliable genetic systems.

## Acknowledgments

I wish to thank D. E. Goldberg for his comments on this paper. Support for this work was provided by a Fulbright-Garcia Robles / CONACYT Fellowship and by the U.S. Air Force Office of Scientific Research under AFOSR Grants F49620-94-1-0103 and F49620-95-1-0338.

## References

1. Adamidis, P. (1994). Review of parallel genetic algorithms bibliography. Technical report, Aristotle University of Thessaloniki, Thessaloniki, Greece.

2. Anderson, E.J. and Ferris, M.C. (1990). A genetic algorithm for the assembly line balancing problem. Technical Report TR 926, Computer Sciences Department, University of Wisconsin-Madison.

3. Baluja, S. (1992). A massively distributed parallel genetic algorithm (mdpGA). Technical Report CMU-CS-92-196R, Carnegie Mellon University.

4. Baluja, S. (1993). Structure and performance of fine-grain parallelism in genetic search. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 155–162, San Mateo, CA: Morgan Kaufmann Publishers.

5. Belding, T. (1995). The distributed genetic algorithm revisited. In D. Eshelmann, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann Publishers.

6. Béssiere, P. and Talbi E.-G. (1991). A parallel genetic algorithm for the graph partitioning problem. In *ACM Int. Conf. on Supercomputing ICS91*, Cologne, Germany.

7. Bianchini, R. and Brown, C. M. (1992). Parallel genetic algorithms on distributed-memory architectures. Technical Report 436, Computer Science Department, University of Rochester.

8. Bianchini, R. and Brown, C. M. (1993). Parallel genetic algorithms on distributed-memory architectures. In S. Atkins and A. S. Wagner, editors, *Transputer Research and Applications 6*, pages 67–82, Amsterdam: IOS Press.

9. Braun, H. (1991). On solving travelling salesman problems by genetic algorithms. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496 of *Lecture Notes in Computer Science*, pages 129–133, Dortmund, Germany: Springer-Verlag, Berlin, Germany.

10. Cantú-Paz E. and Mejía-Olvera, M. (1994). Experimental results in distributed genetic algorithms. In *International Symposium on Applied Corporate Computing*, pages 99–108, Monterrey, Mexico.

11. Chen, R.J., Meyer, R., and Yeckel, J. (1993). A genetic algorithm for diversity minimization and its parallel implementation. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 163–170, San Mateo, CA: Morgan Kaufmann Publishers.

12. Cohoon, J., Hegde, S., Martin, W., and Richards, D. (1987). Punctuated equilibria: a parallel genetic algorithm. In John J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Publishers.

13. Cohoon, J., Martin, W., and Richards, D. (1991a). Genetic algorithms and punctuated equilibria in VLSI. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 134–144, Berlin, Germany: Springer-Verlag.

14. Cohoon, J., Martin, W., and Richards, D. (1991b). A multi-population genetic algorithm for solving the k-partition problem on hyper-cubes. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann Publishers.

15. Davidor, Y. (1991). A naturally ocurring niche & species phenomenon: The model and first results. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 257–263, San Mateo, CA: Morgan Kaufmann Publishers.

16. Dymek, A. (1992). Examination of hybercube implementations of genetic algorithms. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base.

17. Goldberg, D.E. (1991). A comparative analysis of selection schemes used in genetic algorithms. In Gregory Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93, San Mateo, CA: Morgan Kaufmann Publishers.

18. Goldberg, D.E. (1994). Genetic and evolutionary algorithms come of age. *Communications of the ACM*, 37(3):113–119.

19. Goldberg, D.E., Deb, K., and Clark, J. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362.

20. Goldberg, D.E., Kargupta, H., Horn, J., and Cantú-Paz, E. (1995). Critical deme size for serial and parallel genetic algorithms. Technical Report IlliGAL 95002, University of Illinois at Urbana-Champaign.

21. Gordon, V., Whitley, D., and Böhm, A. (1992). Dataflow parallelism in genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature II*, pages 533–542, Amsterdam: North-Holland.

22. Gordon, V. and Whitley, D. (1993). Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183, San Mateo, CA: Morgan Kaufmann Publishers.

23. Gorges-Schleuter, M. (1990). *Genetic algorithms and population structure - A massively parallel algorithm*. PhD thesis, University of Dortmund.

24. Gorges-Schleuter, M. (1992). Comparison of local mating strategies in massively parallel genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature II*, pages 553–562, Amsterdam: North-Holland.

25. Grefenstette, J. (1981). Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt University, Nashville, TN.

26. Grosso, P. (1985). *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model.* PhD thesis, University of Michigan.

27. Gruau, F. (1994). *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Ecole Normale Superieure de Lyon.

28. Holland, J. (1959). A universal computer capable of executing an arbitrary number of subprograms of sub-programs simultaneously. In *Proceedings of the 1959 Eastern Joint Computer Conference.*

29. Holland, J. (1960). Iterative circuit computers. In *Proceedings of the 1960 Western Joint Computer Conference.*

30. Koza, J. and Andre, D. (1995). Parallel genetic programming on a network of transputers. Technical Report CS-TR-95-1542, Stanford University.

31. Kröger, B., Schwenderling, P., and Vornberger, O. (1991). Parallel genetic packing of rectangles. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 160–164, Berlin, Germany: Springer-Verlag.

32. Kröger, B., Schwenderling, P., and Vornberger, O. (1992). Massive parallel genetic packing. In G. L. Reijns and Jian Luo, editors, *Transputing in Numerical and Neural Network Applications*, pages 214–230, Amsterdam: IOS Press.

33. Kröger, B., Schwenderling, P., and Vornberger, O. (1993). Parallel genetic packing on transputers. In J. Stender, editor, *Parallel Genetic Algorithms: Theory and Applications*, pages 151–185, Amsterdam: IOS Press.

34. Kröger, B., and Vornberger, O. (1992). Enumerative vs. genetic optimization: two parallel algorithms for the bin packing problem. In B. Monien and T. Ottamann, editors, *Data structures and efficient algorithms. Final report on the DFG Special Initiative*, pages 330–362. Springer-Verlag, Berlin, Germany.

35. Levine, D. (1994). *A Parallel Genetic Algorithm for the Set Partitioning Problem.* PhD thesis, Illinois Institute of Technology.

36. Li, T. and Mashford, J. (1990). A parallel genetic algorithm for quadratic assignment. In R. A. Ammar, editor, *Proceedings of the ISMM International Conference. Parallel and Distributed Computing and Systems*, pages 391–394, New York, NY: Acta Press, Anaheim, CA.

37. Lin, S-C., Punch, W, and Goodman, E. (1994). Coarse-grain parallel genetic algorithms: Categorization and new approach. In *Sixth IEEE Symposium on Parallel and Distributed Processing*, Los Alamitos, CA: IEEE Computer Society Press.

38. Gorges-Schleuter, M. (1989). ASPARAGOS: An asynchronous parallel genetic optimization strategy. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers.

39. Manderick, B. and Spiessens, P. (1989). Fine-grained parallel genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers.

40. Maresky, J. (1994). On effective communication in distributed genetic algorithms. Master's thesis, Hebrew University of Jerusalem.

41. Marin, F., Trelles-Salazar, O., and Sandoval, F. (1994). Genetic algorithms on lan-message passing architectures using pvm: Application to the routing problem. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 534–543, Berlin, Germany: Springer-Verlag.

42. Merkle, L., Gates, G., Lamont, G., and Pachter, R. (1993). Application of the parallel fast messy genetic algorithm to the protein folding problem. Technical report, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

43. Merkle, L. and Lamont, G. (1993). Comparison of parallel messy genetic algorithm data distribution strategies. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 191–198, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

44. Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers.

45. Mühlenbein, H. (1991). Darwin's continent cycle theory and its simmulation by the prisoner's dilemma. *Complex Systems*, 5:459–478.

46. Mühlenbein, H. (1991). Evolution in time and space – the parallel genetic algorithm. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers.

47. Mühlenbein, H. (1991). Parallel genetic algorithms and neural networks as learning machines. In D. J. Evans, G. R. Joubert, and H. Liddell, editors, *Proceedings of the International Conference Parallel Computing '91*, pages 91–103. Amsterdam: North-Holland.

48. Mühlenbein, H. (1991). Parallel genetic algorithms, population genetics and combinatorial optimization. In J. D. Becker, I. Eisele, and F. W. Mundemann, editors, *Parallelism, Learning, Evolution. Workshop on Evolutionary Models and Strategies and Workshop on Parallel Processing: Logic, Organization and Technology - WOPPLOT 89*, pages 398–406. Berlin, Germany: Springer-Verlag.

49. Munetomo, M., Takai, Y., and Sato, Y. (1993). An efficient migration scheme for subpopulation-based asynchronously parallel genetic algoritms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 649, San Mateo, CA: Morgan Kaufmann Publishers.

50. Muntean, T. and Talbi, E.-G. (1991). A parallel genetic algorithm for process-processors mapping. In M. Durand and El Dabaghi, editors, *Proceedings of the Second Symposium II. High Performance Computing*, pages 71–82, Montpellier, France.

51. Neuhaus, P. (1990). Solving the mapping-problem – experiences with a genetic algorithm. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 165–169, Berlin, Germany: Springer-Verlag.

52. Pettey, C. and Leuze, M. (1989). A theoretical investigation of a parallel genetic algorithm. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers.

53. Pettey, C., Leuze, M., and Grefenstette, J. (1987). A parallel genetic algorithm. In John J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Publishers.

54. Robertson, G. (1987). Parallel implementations of genetic algorithms in a classifier system. In J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*.

55. Schwehm, M. (1992). Implementation of genetic algorithms on various interconnection networks. In M. Valero, E. Onate, M. Jane, J. L. Larriba, and B. Suarez, editors, *Parallel Computing and Transputer Applications*, pages 195–203, Amsterdam: IOS Press.

56. Shapiro, B. and Navetta, J. (1994). A massively parallel genetic algorithm for RNA secondary structure prediction. *The Journal of Supercomputing*, 8:195–207.

57. Spiessens, P. and Manderick, B. (1990). A genetic algorithm for massively parallel computers. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers, Dusseldorf, Germany*, pages 31–36, Amsterdam, Netherlands: North-Holland.

58. Spiessens, P. and Manderick, B. (1991). A massively parallel genetic algorithm: Implementation and first analysis. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann Publishers.

59. Starkweather, T., Whitley, D., and Mathias, K. (1991). Optimization using distributed genetic algorithms. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496 of *Lecture Notes in Computer Science*, pages 176–185, Berlin, Germany: Springer-Verlag.

60. Talbi, E.-G. and Bessière, G. (1991a). A parallel genetic algorithm for the graph partitioning problem. In *Proc. of the International Conference on Supercomputing*, Cologne.

61. Talbi, E.-G. and Bessière, G. (1991b). A parallel genetic algorithm for the graph partitioning problem. In *ACM Int. Conf. on Supercomputing*.

62. Tamaki, H. and Nishikawa, Y. (1992). A parallelled genetic algorithm based on a neighborhood model and its application to job shop scheduling. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature II*, pages 573–582, Amsterdam: North-Holland.

63. Tanese, R. (1987). Parallel genetic algorithms for a hypercube. In John J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Publishers.

64. Tanese, R. (1989a). Distributed genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers.

65. Tanese, R. (1989b). *Distributed Genetic Algorithms for Function Optimization.* PhD thesis, University of Michigan. Computer Science and Engineering.