ABSTRACT
The Genetic Algorithms draw a similarity from the Genetic mutation and Cross Over within populations from biology. The genetic algorithms are highly parallel in nature. These can be used to solve many important problems like Graph Partitioning, Travelling salesman problems, 0-1 Integer linear programming problem etc. When these are implemented, there exists a trade-off between Genetic search qualities and execution performance. In order to improve the execution performance of algorithms, those implementations with lesser communications between populations are considered best. In this direction, we try present an algorithm by discrete small subpopulations. However this implementation reduces the quality of search of the algorithm. Therefore we can improve the quality of search by having a centralized population. In this paper, we review some of the alternatives of implementation of these algorithms on distributed memory architectures in which centralized data can be implemented. We also present an example in which we implement these alternatives of parallel algorithms for predicting the tertiary protein structure. In the final section, we try to provide a performance analysis of the various proposed architectures.


I. INTRODUCTION
Genetic Algorithms is a heuristic search technique which draws its inspiration from the popular “survival of the fittest” principle of natural evolution. First pioneered by John Holland in the 60s, Genetic Algorithms has been widely studied. In genetic algorithms, a collection of possible solutions, known as population is maintained. The search algorithm proceeds in steps called generations. Each generation involves transformation of individual solutions based on their fitness function and results in a new population of solutions. There are two operators of transformation: crossover and mutation. In cross over the pieces of solution are swapped to give new solutions and their fitness is tested. After crossover, mutation is applied which is randomly done to select new characteristics for the solution. The termination is based on either pre specified number of steps or a pre specified level of optimality. GA’s provide alternative methods to solving problem and consistently outperform other traditional methods. Many of the real world problems involved finding optimal parameter may be difficult for traditional methods but ideal for GA’s. Although these are highly deft for solving hard problems, they might take longer execution times. Driven by the need to reduce the execution time, scientists resorted to exploit the parallel nature of GA’s. These came to known as parallel genetic algorithms. The implementation of parallel genetic algorithms involves similar issues as other parallel algorithms do. These include granularity, synchronization and locality of reference. In addition to these, considerations about replication strategy, mutation, distribution of population and quality of search also must be addressed. Here we address the issue of population distribution on distributed memory architectures. The distribution of population, considerably improves the execution
performance. The key is to design independent processes which handle subpopulations of solutions. The implementations with least communications are always considered best. However, there is a compromise with the quality of search in such implementations.

Here, we present certain alternatives for the parallel implementations. These can be characterized as centralized, partially distributed or fully distributed. Also as an example of the applications, we explain the quality of search and the execution performance of our strategies on the problem of predicting tertiary protein structure.

II. IMPLEMENTATION OF PARALLEL GA’S

In this section we discuss some strategies for distribution of population for parallel Genetic Algorithms. We consider synchronized processes in which the processors run for a certain number of generations and then communicate.

Synchronized parallel GA’s can be easily implemented and are efficient in situations where the processors perform sufficient work between communications. The strategies discussed here are for distributed-memory machines, but we also touch upon the application of these on shared memory machines.

- Distributed-memory machine:
  Distributed memory refers to a multiple-processor computer system in which every processor has its own private memory. Computational tasks can operate only on local data, and if remote data is required, the computational task must communicate with one or more remote processors. For this communication some form of interconnection is allowed. It can point-to-point links or switched network of processors. The links between processors can follow standard protocol.

- Shared memory machines:
  In shared memory machines, a number of multiprocessors have an access to a large memory. The processors need not be aware of where the data resides. However there are performance penalties and race conditions that need to be avoided in these implementations.

III. PARALLEL GA’S ON DISTRIBUTED MEMORY ARCHITECTURES

Genetic Algorithms are highly parallel in nature. Thus, it fairly simple to implement these on distributed memory machines. However, the distribution results in denigration of the quality of search results obtained in contrast to those obtained through centralized implementations. The trade-off between search quality and distribution can be resolved through semi-distributed implementations. In these all the processors are have their private memory. In addition to these, the processors are grouped together in order to form clusters. Each cluster has its own large shared memory. This strategy exhibits least contention and communication overhead. It may, although, suffer from execution overhead. It still draws the benefits of centralized implementation and result in obtaining better search results in fewer generations. In the subsequent sections we describe the different implementation strategies we studied.

1. Covenant Implementation
One of the extremes in distribution of population is a single large population. The covenant implementation is intended to achieve this extreme. In this strategy, as the name suggests, an agreement between the processors exists. One of them is the kernel processor and the other act as slaves. The processors are arranged in star topology. Only the master has the power to access the memory. It collects information from the memory and executes the replication algorithm. It then sends the couple solutions to the slaves. The slaves are responsible for transforming them using the genetic operators and then test their validity using the fitness functions. The new individuals are then returned to the master processor. This master-slave representation has some serious challenges. These are large amount of communication, a fraction is sequential computation (replication) and the granularity is too fine. All these problems degrade the execution performance of these algorithms. We can solve these problems by making the slaves work on several generations, thus reducing the granularity and the amount of communication. Another flaw with this strategy is that the master remains idle, while the slaves are working. This can be resolved by making the master work on a subset of population immediately after sending couples to slaves. After master is done with his work, the slaves can send back subpopulations to master. And the process continues like a handshake.

2. Semi-Disparate implementations
This strategy is aimed at settling for an intermediate level of distribution of population. It draws benefits from the global knowledge of current population and its distribution. In this strategy, there is no single kernel processor. Instead, we have clusters of processors, each having its own kernel. All the kernels of the processors are arranged in a torus topology. That is, each kernel has a neighbor to its north, south, east and west. Every cluster works individually on its subset of population as if in a covenant environment. At specified intervals of time, all the kernels communicate with each other and exchange some of their best individuals. The communication is synchronized, so that there is no contention in the network. This implementation reduces the overhead of maintaining a single master. However, this also inherently suffers from the problem of large amount of inter-processor communication and semi distributed replication.

3. Disparate implementations
In disparate implementations, all nodes work independently of each other. Every node has the total population of solutions and runs the complete sequential GA on it independently. This organization is free from any shared populations. All the nodes are connected in torus topology. Communication amongst processors happens periodically and they send their best individuals across the network. Communication happens during initialization (to get a description of problem to be solved) and termination (in which all processors send their best individuals to the processor responsible for displaying the results). This implementation has two main advantages. Firstly, there is no contention among processors and all have their own population set. Secondly, the amount of communication can be made as minimum as desired. A problem with this implementation is that a lot of computation may be done on locally fit solutions. However, these may be globally
unfit and finally discarded. This way a lot of computation time gets wasted.

4. Totally disparate implementations
This is the other extreme followed in the distribution of population. In this strategy, the processors don’t exchange their individuals. Communication takes place only at initialization and termination of the algorithm. Thus, there is absolutely no contention and the communication is even lesser.

IV. A PARALLEL GENETIC ALGORITHM TO PREDICT THE TERTIARY PROTEIN STRUCTURE
Proteins are complex molecules that can be broken down into sequences of amino acids bound by a peptide bond, which play a fundamental role in nearly all biological processes e.g. immune response mechanisms, enzymatic activity, signal transduction etc. Tertiary structure is a 3 dimensional, native structure of a single polypeptide or protein, arranged in such a way that the hydrophobic side chains are held interior and the hydrophilic groups are held outside, imparting stability to the molecule.

Tertiary structure assembles the different secondary structure components by describing the folding of the polypeptide chain in a protein structure arrangement. The domain is the unit of tertiary structure. Multi-domain proteins, which can be built from several domains, represents their arrangement within each chain and relative to each other. These can also be divided into several classes on the basis of size, their physical and chemical properties. The main classification is into hydrophobic residues, owing to its weak interaction to solvent water molecules, and hydrophilic residues, due to its ability to form hydrogen bonds with water. Each amino acid consists of a common main-chain part, containing the atoms N, C, O, Ca, two hydrogen atoms and a specific side chain. The amino acids are joined by the peptide bond, the planar CO–NH group. The main degrees of freedom in forming the 3D trace of the polypeptide chain are the two dihedral angles, x and y on each side of the Ca atom. Owing to steric restrictions, these angles can have values only in specific domains in the x, y space. The side chains branch out of the main chain from the Ca atom and have additional degrees of freedom enabling them to adjust their local conformation to their environment. The importance of the protein structure prediction lies in its wide presence in living organisms, applications of computational protein structure prediction directed to computer assisted drug design and computer assisted molecular design. The use of a near-optimal metaheuristic, such as a Genetic Algorithm, is one of the most promising optimization methods as it explores minimal number of potential structures. A powerful exploration of the conformational space can be achieved by using these population-based metaheuristics. Given the primary structure of a specified protein, we need to determine its ground state conformation. However, they have limited search intensification capabilities.

V. RELATED WORK
Two search methods have traditionally been employed to predict the tertiary structure of a protein from its primary structure: Molecular Dynamics (MD) and Monte Carlo (MC).

Molecular Dynamics takes into account the system’s reaction to the forces the atoms exert on each other, assuming that atoms move in a Newtonian manner. While MD methods are based on the direct simulation of the natural folding process, MC algorithms are based on minimization of an energy function, through a path that does not necessarily follow the natural folding pathway. MC calculates the free energy of successive small conformational step, which is accepted if the free energy is reduced compared to the previous conformation. While MD methods almost by definition require a full atomic model of the protein and detailed energy function, MC methods can be used both on detailed models or on simplified models of proteins.

VI. PROPOSED GENETIC ALGORITHM APPROACH TO THE PROTEIN TERTIARY STRUCTURE PREDICTION PROBLEM:

MC algorithms, though efficient, are more likely to get caught in a local minimum. Thus, using GAs to address the protein folding problem may be more effective.

When folding a chain with a MC algorithm, which is based typically on changing a single amino acid, it is common to get into a situation where every single change is rejected because of a significant increase in free energy, and only a simultaneous change of several angles might enable further energy minimization. This kind of simultaneous change is provided naturally by the crossover operator of GA.

> COMPUTER REPRESENTATION:

Owing to a largely fixed geometry of the protein, the main degrees of freedom in determining its three-dimensional conformation are the two dihedral angles \( x \) and \( y \) on each side of the \( C_a \) atom. Thus, a protein can be represented as a set of pairs of values for these angles along the main chain \([x_{1},y_{1}],(x_{2},y_{2}), (x_{3},y_{3}), \ldots, (x_{n},y_{n})\] which can be represented in Cartesian coordinates. This representation has the advantage of being easily converted to and from the conformation of a protein. However, it would mostly create invalid instances as a result of use of the mutation operator being applied to the population, where atoms either collide or are too far apart. It does not contain a mechanism that can ensure that the encoded structure is free of collisions, i.e. that the dihedral angles do not describe a trajectory that leads one atom to collide with another atom along the chain. However, a filter can be used to eliminate the invalid individuals but would considerably slow down the process.

> OPERATORS

To implement a genetic algorithm, it is necessary to encode the variables of the optimization problem into the genes. The genes of the parents are then operated on through recombination and mutation to produce the genes of the children.

1. Selection

The selection operator uses a fitness function to identify the test individuals of the current population that will serve as parents of the next generation. The fitness value of each individual is given by a problem-specific function. The
selection mechanism ensures that the best individuals have a higher probability to be selected to reproduce to form a new generation.

1.1 Fitness Function
Numerous energy functions have been used as part of the various GA based protein structure prediction protocols, ranging from the hydrophobic potential in the simple HP lattice model to energy models such as CHARMM (Chemistry at Harvard Molecular Mechanics), based on complete, detailed molecular mechanics. The fitness function can be easily modified to include terms that are not used in the traditional methods of protein structure prediction. We have used a simple steric function as the fitness function.

2. Crossover
Crossover randomly selects pairs of crossing points and exchanges substrings between them to produce new off-springs. The primary exploration mechanism for GAs is crossover. An example of crossover. A branch is selected for crossover in each of the parent trees. In this example, the first left branch from the first parent is exchanged with the first right branch of the second parent. This creates two new trees.

3. Mutation
The mutation operator is usually considered a secondary operator, used mainly 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.

VII. ALGORITHM FOR PROTEIN STRUCTURE PREDICTION
Algorithm_PSP_GA()
1: Select Initial Population G(0)
2: \( t \leftarrow 0 \);
3 repeat while \( n(G(t)) > 1 \) (Number of offspring’s >1)
4: Evaluate(P(t));
5: Apply fitness function on G(t)
   \[ G(t) \leftarrow \text{Selection}(G(t)) \];
6: Alter between the various reproduction operators and apply the chosen operator to obtain the new population G(t+1)
   \[ G(t+1) \leftarrow \text{Apply Reproduction Operator}(G(t)) \];
7: G(t + 1) \leftarrow G(t)
8 \( t := t + 1; \)
9: End loop in Step 3

In this algorithm, reproduction operators are applied at the initial population to yield the first set of offsprings. Every new set of offsprings is then iteratively tested by the fitness function (Step 5) and if the offsprings are closer to the solution, they undergo further reproduction operation (Step 6). Note that any of the reproduction operators may be applied to the offsprings. The offsprings \( G(t+1) \) then serves as our new population \( G(t) \) (Steps 7 & 8). A problem specific fitness function must be applied for evaluation.

**VIII. PERFORMANCE ANALYSIS**

We reviewed the performance of the above algorithm on each of the proposed architectures and plotted the results as a graph between the running time and the number of processors used in the model. It can be seen from the above graph that the running time in a disparate implementation is greatly reduced, and thus, the performance of this implementation is better than a covenant implementation. However there exists a tradeoff between performance and search quality. The choice of the architecture is dependent in the application. If we are using the GA to predict a critical problem where search quality is a crucial factor we prefer a covenant implementation despite its slow running time.

**IX. TRADE OFF BETWEEN APPLICATION IN COVENANT AND DISPARATE IMPLEMENTATIONS OF PARALLEL GENETIC ALGORITHM:**

The disparate implementation of the algorithm is better suited to the protein structure prediction as the population is large and its running on a covenant system will take much more time. Both the semi-disparate and disparate implementations show much lesser time complexity and an analysis leads to the result that the semi-disparate is more effective. Thus using a parallel genetic algorithm on distributed memory architecture is the best possible alternative, running in lesser time and yielding better results.

**X. CONCLUSION**

The protein structure prediction problem is especially suited for the GA approach, as GA methods have produced significantly better results than the MC method. The ability of the parallel genetic algorithms to describe many biological processes comes from its unique ability to model cooperative pathways, which makes it better suited to predict protein structure. The disparate and semi disparate implementations have proved to be more effective and of better time complexity, hence making the protein structure prediction even better suited for distributed memory architecture. Processes that approach an
optimum value in an iterative manner can be modeled on parallel GAs. Protein structure prediction serves as an excellent example, and remains an open research field due to complexity matters, especially in multiple domains.

REFERENCES