CENG 491 Computer Engineering Design Requirement Analysis Report

YAKUT

Sefa Kılıç
Işıl Doğa Yakut
Yunus Başağalar
Yiğit Çağrı Akkaya

Contents

1 Introduction

1.1 Purpose

Throughout this document, we will introduce the needs of the project of parallelization of genetic algorithm library, GAlib, in detail for the users. Target users are people who not only use genetic algorithms as a solution to optimization problems, but also need better performance in speed and reliability.

1.2 Scope

The created parallel library, at the end of the project, will be called **PGAlib (Parallel Genetic Algorithm library)**. Library will provide improvement in performance when executed on parallel computers whereas on single processor computers, performance will remain the same. The goal of the project is to reach a reasonable increase in execution time with respect to number of processors.

1.3 Definitions,Acronyms and Abbreviations

- GA : Genetic Algorithms
- Genome : A set of parameters, often represented as a simple string, although wide variety of data structures is used.
- Fitness function : A type of objective function, the computes the optimality of a solution.
- Population : A set of genomes to be given to a genetic algorithm
- Crossover : A genetic operator where given two individuals of a population exchange a part of their genome with each other.
- Mutation : A genetic operator to maintain diversity, an arbitrary bit of a genome changes state.
- Parallelization : Distributing the process into several processor to work in parallel. Data parallelization and task parallelization main type of distributions.
- MPI : A message passing interface among different machines on a cluster.

1.4 References

[1] Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers, B.WILKINSON and M.ALLEN [1999] by Prentice-Hall
[2] A Survey of Parallel Genetic Algorithms , Erick Cantu-Paz [1998] , Paris
[3] D Beasley, RR Martin, DR Bull An overview of Genetic Algorithms, Part1 Fundamentals, 1993
[4] D Beasley, RR Martin, DR Bull An overview of Genetic Algorithms, Part2 Research Topics, 1993
[5] M Wall - GAlib: A C++ Library of Genetic Algorithm Components Department, Massachusetts Institute of Technology, Aug, 1996

2 The Overall Description of the Project

2.1 Product Perspective

PGAlib is planned to be strongly dependent on GAlib. GAlib will form the constraints and specifications of our project. Similar projects focus on different specific aspects of GAlib and the target systems to operate on are not generic, such an example is MPIGA. Our project will focus on parallelizing GAlib entirely and will be built in such a way that it does not require a specific machine architecture. PGAlib will not depend on another system to work, thus it will be build to work as a whole component.

2.2 Project definition

Optimization problems arise in many different fields of study and genetic algorithms are known to be powerful search techniques especially when applied to this kind of problems. This project is focused on getting better or similar results in a shorter time period by combining cluster computing and genetic algorithms.

We choose the genetic algorithm library GAlib after a research on available genetic algorithm libraries with the assistance of Dr. Onur Tolga Sehitoglu. Its widespread use and ease to manage structure is an advantage when parallelizing a library since parallelization adds to degree of complexity.

In this project, we aim to produce an efficient, high performance and reliable parallel genetic algorithm library. It will be able to work on computer clusters given the required software products. Also in the absence of a cluster computer it should run same as with

GAlib. Another aspect of GAlib is its wholeness, we plan to parallelize GAlib entirely since it is a intact library, which runs on a main loop until termination, thus makes it impossible to partly parallelize it. So the user-end will have no difference from GAlib itself. Like GAlib our library will have similar characteristics such as letting user define custom evaluation function and migration method.

2.3 Goal of the Project

- Developing a complete parallel library with a base library.
- Using computer cluster tool to aid our goal.
- Fault-detection and fault-removal
- Optimizing performance scaled to parallelization achieved.
- Create an easily usable parallel library

3 Literature Survey

3.1 Overview of Genetic Algorithms

Genetic Algorithms is a method to find solutions to optimization problems which do not require an exact solution. It imitates the the evolution with its selection and reproducing techniques such as crossover, mutation and elimination of the individuals which are not fit to environment. It operates on a given population, where population is defined to be a set of abstract representations of individuals' genome. In initialization of population a random combination is assigned to individuals. Then there are two phases before a new population is formed. First individuals are evaluated by a predefined fitness function, usually implementation of fitness function can differ according to the needs of the user. Then by several different reproduction methods a new population is generated. The most common two reproduction methods are mutation and crossover. Mutation requires only one individual and the process change the combination of representation. Second crossover is exchanging a part of the genome between two individuals. After initialization, these two phases are repeated until a termination condition reached. Above is a chart that show the process of a common genetic algorithm.

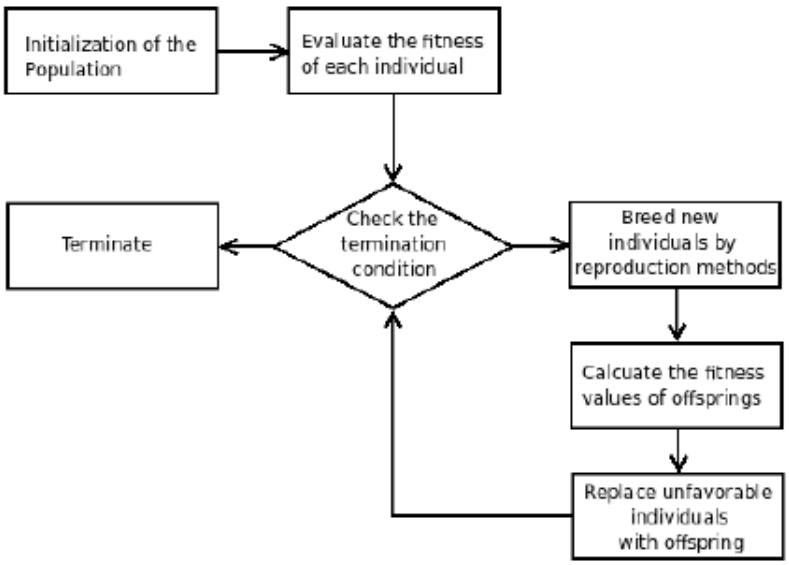

Figure 1: Flowchart for genetic algorithm

3.2 Implementations of Genetic Algorithms

Below are several developed genetic algorithm libraries categorized according to their base programming languages.

3.2.1 C++

EO Evolutionary Computation Framework

EO is a open development library and is distributed under GNU Lesser General Public License. It is started by Geneura Team at University of Granada. It is a template based, C++ evolutionary computation library which covers most of the evolutionary computation techniques, by defining them as classes. It is easy to implement your own class with given template files. It's class structure is component based in order to construct a class which is not defined in the library, from existing abstract or concrete classes. It works on both Windows and Unix operating systems. There are several applets ParadisEO and DegaX which provide flexible design, meta heuristics for multi objective optimization and an ActiveX control.

GAlib

GAlib is a work of Matthew Wall using MIT resources. The library is an effective tool to optimize in any C++ program with any representation, which makes it advantageous compared with other implementations. Customization of Galib classes is also available. It is easy to try different objective functions, representations, genetic operations and algorithms. In addition to the basic built-in types, GAlib provides the components that may be needed when designing a new genetic algorithm classes.

GAGS

GAGS is a C++ class library which contains classes used to program all the elements that constitute a genetic algorithm. As such, it includes classes for chromosomes, genetic operators, views and population. Genetic operators defined in GAGS are not limited to mutation and crossover, it includes 9 predefined genetic operators. Some of these are bit-flip mutation, creep, crossover, remove/reinsert, transpose, kill, remove, randomAdd. Also, evaluation or fitness function can be any C / C++ function

3.2.2 Java

GAJIT

GAJIT is ported from GAGS, to implement it in Java. During this transportation, change in the classes occurred due to different structures Java and C++. It is developed for a special purpose for a project by a person, which takes it far from a generic library. Though it is a good library which is under Java environment. It does not add any features to GAGS, though it is a bit different since it is developed for a specific use.

GA Playground

GA Playground is a toolkit for user to run your optimization problem. User is only required to have basic programming skills in order to write the fitness function. Although different aspect of the toolkit can be altered, its main problem is solving an optimization problem, which limits its flexibility and generic content in its context. It is available in both application and applet form, but it can be categorized as an application due to its requirement of recompiling of at least one class. Another aspect is that it can take advantage of the Java cross-platform nature, which lets people who are interested in Genetic Algorithms come across with GA Playground.

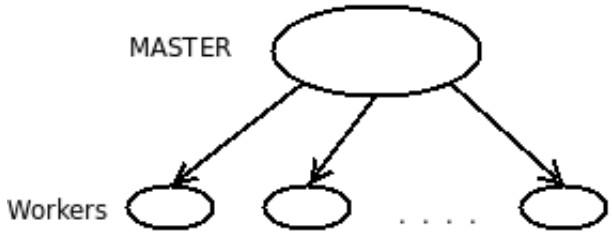

Figure 2: Single population master-slave GA

3.2.3 ANSI C

PGAPack

PGAPack is one of the few implementations of parallel genetic algorithms. Its main purpose is to provide an integrated and portable genetic algorithm library. Its compatible with C and Fortran and can run on not only parallel computers but also single processor computers and can support custom data types.

GAUL

GAUL is another assistant to problems that require genetic algorithm implementation and similar to EO is released under GNU General Public License. GAUL differs from other library by its different approaches to genetic algorithms. In its implementation it supports three evolutionary schemes, which are Darwinian, Lamarckian and Baldwinian approaches. Also other optimization algorithms are built-in such as hill climbing, tabu search, simulated annealing, etc. Also supports multiprocessor implementations on OpenMP 2.0, MPI, forked process model and pthreads.

3.3 Parallel Genetic Algorithms

We studied parallel genetic algorithms from several sources but the most comprehensive but concise was a thesis by Erick Cant-Paz. The main idea behind parallelization is to divide into chunks and solve these chunks in parallel. This divide and conquer can be applied Genetic algorithms successfully. Three main types of Parallel Genetic Algorithms are global single-population master-slave GA s (Figure 2), single-population fine-grained GA (Figure 3) and multiple-population coarse grained GAs (Figure 4). In single-population master-slave GA there is a single population and parallelization is done on the evaluation

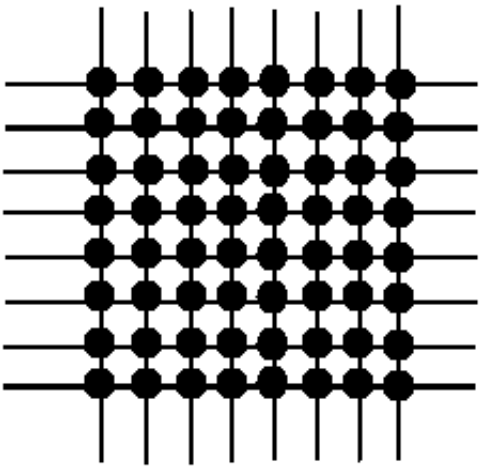

Figure 3: Single population fine-grained GA

level. For the evaluation/fitness function the individuals are assigned to processors (slaves), and run in parallel, where after the evaluation function is done the information is gathered back at the master processor.

In multiple population parallel GAS the subpopulations on different machines form the population and the algorithm is applied separately. Which causes the populations where the functions are executed rather small, thus converge faster. But the overall result might be poorer, the effects of this proportion difference is still not known exactly. Another issue in this kind of parallelization is that a concept called migration exists. Migration is exchanging of individuals between subpopulations occasionally, the control of this method is done by several parameters and its effects are also exact. The last type of parallelization is hierarchical parallel algorithms which combine the previous two types their degree of complexity is very high. Most of these implementation have multiple population GAs at the upper level. Some hybrids have fine-grained GAs at the lower level (Figure 5). Also a mixed parallel GA exists.

3.4 Existing Parallel Genetic Algorithm Libraries

3.4.1 MPI Parallelization

PGAPack is one of the best examples of parallelization of genetic algorithms by using MPI. PGAPack is a general purpose, data structure-neutral, parallel genetic algorithm library. It is intended to provide most capabilities desired in a genetic algorithm library, in an

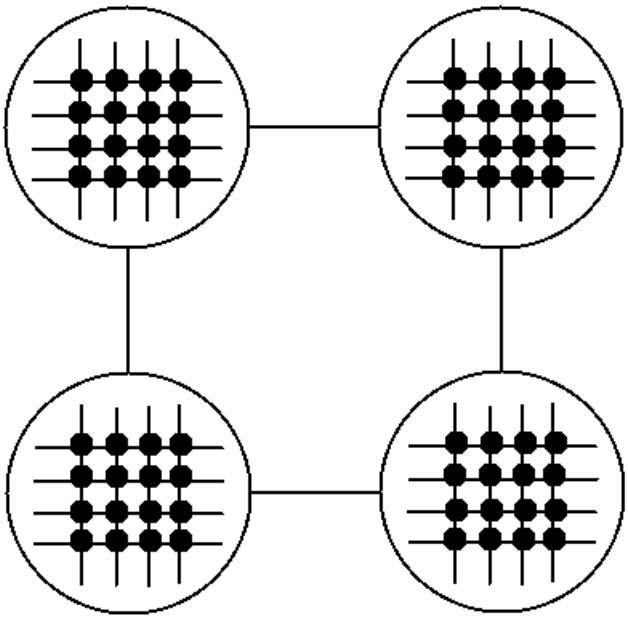

Figure 4: Multiple population GA at the upper level

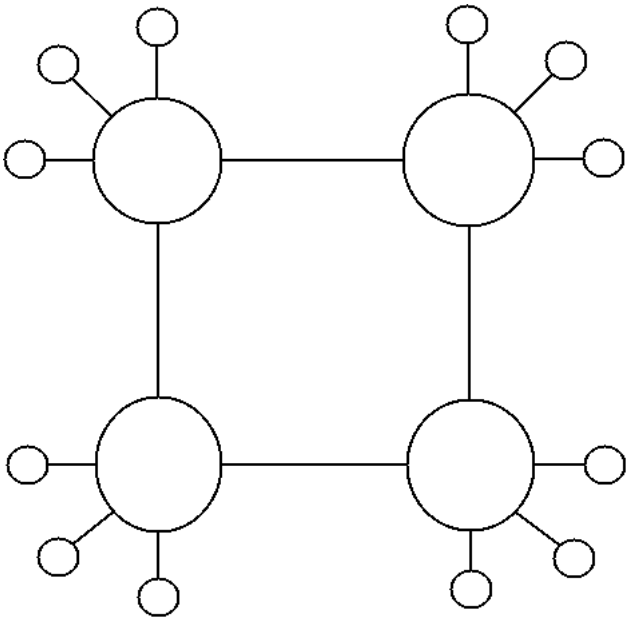

Figure 5: Multiple population GA at the lower level

integrated, seamless and portable manner. PGAPack supports for multiple data types; portability across uniprocessors, multiple processors and workstation networks; Fortran and C interfaces; multiple genetic algorithm operators and parameter choices. PGAPack is implemented by using the MPI standards. Implementations of MPI exist for both sequential(uniprocessor) and parallel(multiprocessors, multicomputers and workstation networks) computer hardware, thereby allowing PGAPack to run on all these machines without any code changes.

3.4.2 PVM Parallelization

Parallel island model is the best suitable model on parallelization of genetic algorithm library with PVM. The island model is designed to exploit a coarse grained architecture. Each processor is given a population of individuals. The processors evolve their populations using a serial genetic algorithm. Periodically a processor may migrate a number of its individuals to another population. The amount of communication involved in the island model appears to be much more manageable than other models. Depending on the factors above, parallelization of genetic algorithm library has been made using PVM.

4 Requirements

4.1 Detailed Analysis of Requirements

Below is a detailed description of what we plan to achieve during our development process. These results are obtained by examining the overall process loop of GAlib.

- **Customization of Evaluation Function**

  In GAlib each genome can have a unique evaluation function defined by the user. In contrast population evaluation function is default. Defining a unique function for each genome is achieved in two steps. First the user must define evaluation functions by implementing them, this part requires basic programming skills. Then when defining the genomes of a population, evaluation functions are passed to the genomes constructor. As can be seen interior of evaluation function can not be altered by the parallelization progress unless there are parallel functions called by evaluation function already. Thus parallelization of the evaluation is done by viewing

the evaluation function as a black box. For instance, parallelization of evaluation function can be achieved by assigning genomes to processors and then calling the evaluation function of the specific genome in the step function.

- **Distribution of Genomes over Processors**

  We have two options when distributing genomes over processors. First choice is to distribute equal number of genomes to each genome or we can request genomes from processors that have the functionality of a pool of data. If the evaluation function is uniform over genomes it is advantage to equally distribute them since the function of each genome will finish at about the same time(see Figure 6), thus the processor do not need additional genomes to execute when the current job is finished(see Figure 7). However if the uniformness is not certain second option can be preferred. Where the idle processors do not sit and wait but request another genome to execute evaluation function.

- **Parallelization of Population**

  The main section where parallelization will be implemented is populations. For convenience we can think population as an array of genome pointers where each genome has its own evaluator and initializer besides the population initializer and evaluator. We will provide a default initializer and an evaluator function to each population. These implementations of the functions will be the same as built-in initialer and evaluator of the GAlib itself. The user can implement his own functions with the mentioned customization options. Though if the evaluation function of the population is customized, parallelization is not achievable since the evaluation function might require additional information from genomes where with parallelization this information can not be retrieved. In the case that default evaluation function of population is used, thus executing evaluation function of each genome sequentially we can use the algorithms of distribution of genomes over processors. The decision of whether using the first or the second method depends on the type of the genetic algorithm. Each genetic algorithm will be implemented as parallel according to its characteristics. Depending on the type of the genetic algorithm the usage of OpenMP and MPI will also differ.

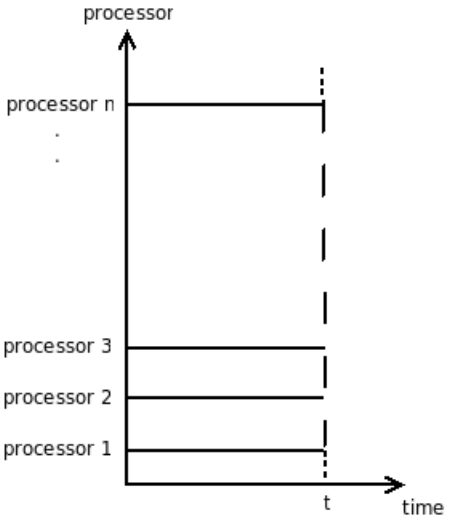

Figure 6: Uniform Evaluation Time

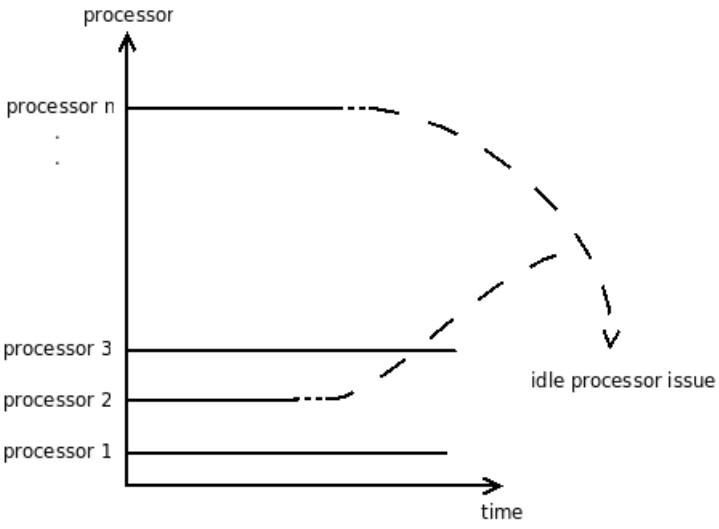

Figure 7: Non-Uniform Evaluation Time

Another issue on parallelization of population is generating new population out of the old one. To form an offspring, first a father and a mother must be selected from the current population. After that selected parents will be assigned to different processors to mate. Crossing over is applied in the process of mating. Then mutation occurs for the offspring which is again executed in parallel. There are two drawback at this point. First, two options arise when distributing parents to processors. First option is to assign couples right after they are selected from the current population, second one is to assign parents after all of them are formed. Our second concern is when returning an offspring from an assigned processor, should we wait until all of the offspring is generated or should each offspring be returned right after its generation. At a first glance sending information bits by bits, returning individually, increase communication overhead, although when using the second type of distribution of genomes the advantage of using idle processors can change interpretation of the situation.

- **Parallelization of Genetic Algorithms**

GAlib consists of four different classes for four genetic algorithms. These four GA types are Simple, SteadyState, Incremental and Deme. First three of them are executed on a single-population thus there is no migration defined in these algorithms. Thus, dividing a given population into several subpopulations will effect the reliability of the algorithm since they will not effect each other during the whole process. So we have two options left: first option is to parallelize only the evolution and selection functions. In this first option unless the size of the population is large enough we are not taking advantage of the HPC's power. Second option is to implement our own migration policies on these three algorithms. By this implementation of migration subpopulation can continue on to have effect between each other, thus we will be able to make better use of HPC's power without sacrificing of reliability. However since fourth algorithm,Deme, uses migration on multiple population, thus an implementation of the second option already exists we will prefer first option which is to parallelize only the evaluation(see Figure 8) and selection functions. For the fourth and last algorithms migration function is left to user to be defined, which restricts us to not being able to parallelize the migration functions. Though ,the major aspect of the migration concept with respect to performance is the selection of the closest nodes which the migration will occur. For migration we plan to implement different

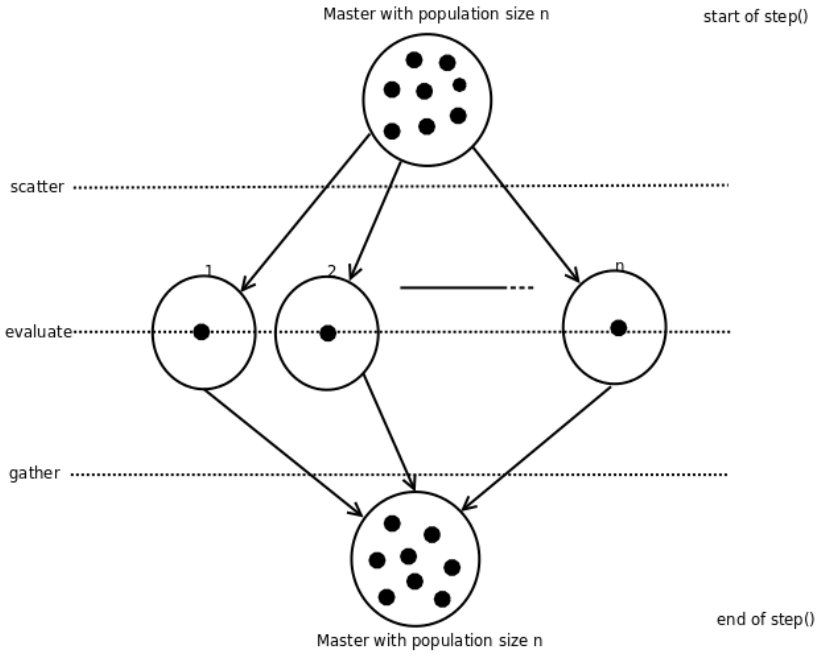

Figure 8: Parallelization of evaluation function of Simple,SteadyState and Incremental Genetic Algorithms

migration algorithms for different topologies in order to give user several options to choose.

Only class that we have not discussed is the genome class, which does not effect the process of parallelization. Instead we mentioned several other parallelization options.

4.2 User Interface

Since the issue of compatibility of PGAlib with GAlib is essential for the user, it is also a significant issue in our project. Our goal is to keep the changes to a minimum on the inner working of GAlib.

- Generate a genome or a population. There will be no additional genome class. User can use genome types same as GAlib genome classes. Thus size information and objective functions will be adequate for construction of genome. And a sample genome and size of population is mandatory for construction of population.
- Construct an implemented genetic algorithm in PGAlib classes, with a genome or a population given as an argument.

- New generations can be generated a step at a time by step functions of algorithms or direct solution can be computed through evolution function. As another option user can define his own operator just as in GAlib.

In conclusion parallelization is not defined by the user-end and implemented by our team. Thus user will be able use PGAlib without any further information on parallel computing and clusters.

4.3 Functional Requirements

4.3.1 Hardware Requirements

Multi-Processor

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently. Main advantage of the parallel computing is speeding up the execution time. Parallel computing uses multiple processing elements to solve a problem. This accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be single computer with multiple processors, a set of network connected computers or combination of these. In our project, a computer with multiple processors will be used in order to get advantages of the parallel computing.

Shared Memory

In computing, shared memory is a memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies. In computer hardware, shared memory refers to large block of random access memory that can be accessed by several different central process units in a multi-processor computer system. A shared memory is relatively easy to program since all processors share a single view of data and communication between processors can be as fast as memory accesses to same location. In order to use OpenMP, which is an application program interface that supports multi-platform shared memory multiprocessing programming in C/C++ and Fortran on many architectures, hardware of our projects needs a shared memory.

InfiniBand

InfiniBand is a switched fabric communications link primarily used in high-performance computing. Its features include quality of service and fail over, and it is designed to be scalable. Like Fibre Channel, PCI Express, Serial ATA and many other modern interconnects, InfiniBand is a point-to-point bidirectional serial link intended for the connection of the processors with high speed peripherals. The serial connection's signaling rate is 2.5gigabit per second in each direction per connection. The single data rate chips have a latency of 200nanoseconds, and DDR switch chips have a latency of 140nanoseconds. InfiniBand uses a switched fabric topology, as opposed to a hierarchical switched network like Ethernet. To avoid large communication overhead, InfiniBand meets our hardware expectations.

4.3.2 Software Requirements

GCC

C++[1] is a general-purpose programming language. It was developed by Bjarne Stroustrup in 1979 at AT&T Bell Labs. It is a direct descendant of C that retains almost all of C as a subset. It supports data abstraction, object-oriented programming and generic programming. Classes, virtual functions, operator overloading, multiple inheritance, templates and exception handling are other main features of C++.

The GNU Compiler Collection (GCC) [2] is a compiler system produced by GNU project. It supports various programming languages. Originally it was a C compiler, but in 1987 it was extended to compile C++. It has also OpenMP, popular parallel language extension support.

GAlib was written in C++ which restricts us to use C++ in our project for compatibility purpose.

OpenMP

The OpenMP[3] is an application programming interface. It provides a portable, scalable model for shared memory parallel applications. It supports C/C++ and Fortran on UNIX and Windows NT. It is defined and endorsed by a group of major computer hardware and software vendors.

[1] http://www.research.att.com/ bs/C++.html
[2] http://gcc.gnu.org/
[3] http://openmp.org

MPI

MPI is a library standard used to program parallel computers. It is not a programming language. There are implementations of MPI libraries for C, C++ and Fortran. It is a standard for communication among processes on a distributed memory system. MVAPICH and OpenMPI are some implementations of MPI.

GAlib

GAlib[4] is a library of genetic algorithm components written in C++. It has tools for using genetic algorithms for optimization problems using any representation and genetic operators. It was implemented by Matthew Wall.

4.4 Non-Functional Requirements

Scalability

Our product will be horizontally scalable. It will protect the performance on a system in case of adding new nodes to the system.

Usability

GAlib helps the user on definition of a representation and genetic operators. In many cases, user can use the built-in representation and operators with little or no modification. User can also extend the capabilities of built-in objects by deriving new classes or defining new functions. In our project we will preserve these properties of GAlib while parallelizing it.

Efficiency

Since we have multi-processor parallel computer which provides us large number of processors and large memory size, it might seem like we do not have constraints. Although if we use more resources than we need, the efficiency will be reduced which is actually the main goal of this project. Thus efficiency is a vital issue to us. We will handle efficiency problems carefully and throughly.

[4]http://lancet.mit.edu/ga/

Quality

The quality measure of the project is not only about correctness and completeness of the final product but also is about the integrity, orderliness of the whole development process. Thus we will try to build the library which is error-free as much as possible.

Open Source

PGAlib project is an open-source project. All source codes, documentation and other materials will be accessible for everyone during the development and at the end of the project.

Documentation

We plan to provide a detailed end-user documentation for a better understanding of usage of PGAlib. This documentation will include changes and additions done for parallelization. In addition to user documentations,as a part of software development process, technical documentation will be done such as initial design report, detailed design report and weekly progress reports.

5 Schedule of the Project

The general view of the schedule of the project until the end of the first semester is as follows. Figure 10 shows project schedule until requirement analysis report milestone. Figure 11 shows the rest of the semester.
The schedule table of next semester consist of coding, debugging and performance analysis tasks.

Task Name	Days	Start	Finish
Research Phase	**30**	**29.09.08**	**07.11.08**
Topic Proposal	5	06.10.08	10.10.08
Introduction to cluster computing	5	14.10.08	20.10.08
Parallel programming	5	13.10.08	17.10.08
MPI	11	17.10.08	31.10.08
OpenMP	11	17.10.08	31.10.08
Introduction to genetic algorithms	5	29.09.08	03.10.08
GAlib	7	30.10.08	07.11.08
Other parallel genetic algorithm libs	3	05.11.08	07.11.08
Analysis Phase	**6**	**07.11.08**	**14.11.08**
Requirement Analysis	5	10.11.08	14.11.08
Requirement Analysis Report	0	14.11.08	14.11.08
Develop Project Plan	3	07.11.08	11.11.08
Design Phase	**45**	**17.11.08**	**16.01.09**
Initial Design	15	17.11.08	05.12.08
Initial Design Report	0	05.12.08	05.12.08
Synchronization	4	18.11.08	21.11.08
Communication	5	24.11.08	28.11.08
Load Balancing	6	01.12.08	08.12.08
Detailed Design	28	09.12.08	15.01.09
Detailed Design Report	0	16.01.09	16.01.09
Final Phase	**20**	**29.12.08**	**23.01.09**
Prototype	20	29.12.08	23.01.09
Final Design Analysis	8	14.01.09	23.01.09
Demonstration	0	23.01.09	23.01.09

Figure 9: Task List

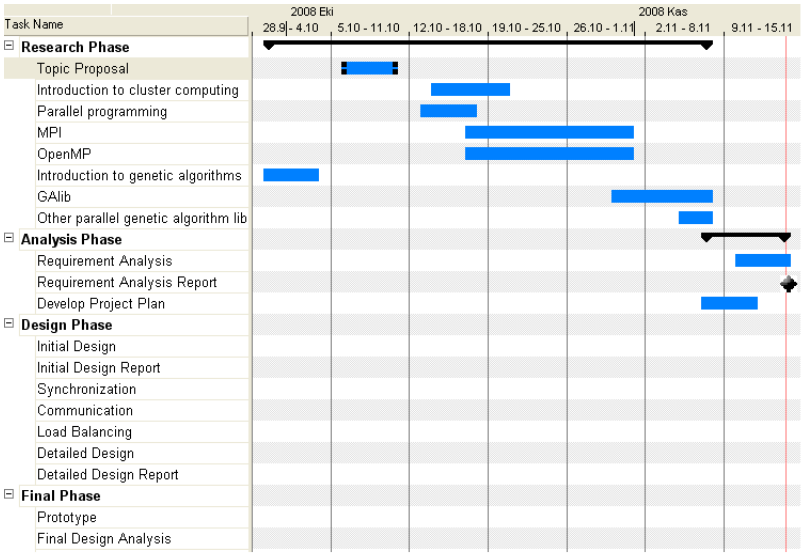

Figure 10:

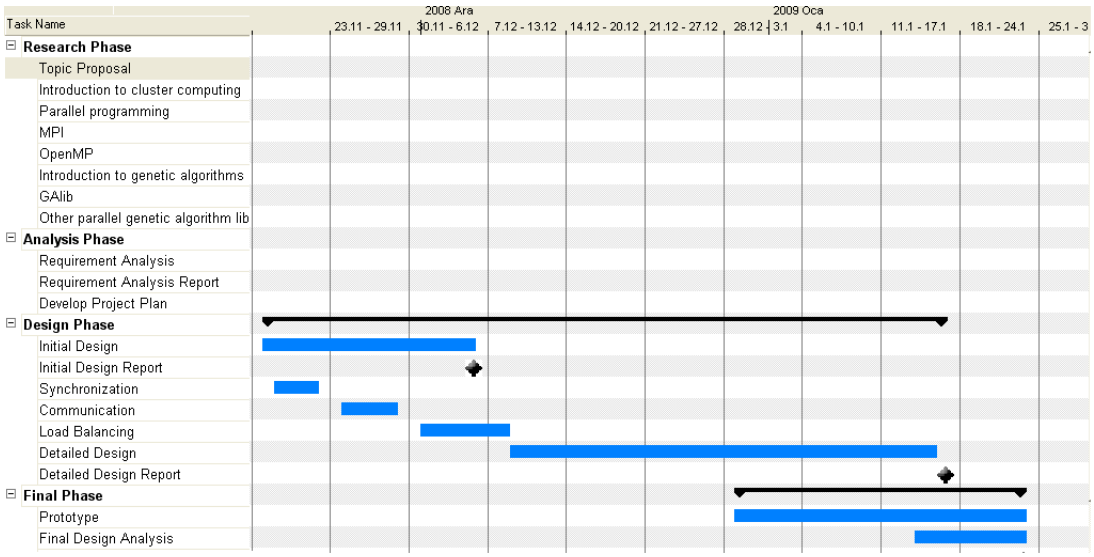

Figure 11: