PARALLEL IMPLEMENTATION OF A FORECASTING ALGORITHM

By

SREENIVASA SIVA BHANODHAY NANUGONDA

Bachelor of Technology in Electronics and

Communication Engineering

Jawaharlal Nehru Technological University

Hyderabad, Andhra Pradesh, India

2009 - 2013

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2015

PARALLEL IMPLEMENTATION OF A

FORECASTING ALGORITHM

Thesis Approved:

Dr. Christopher John Crick

Thesis Adviser

Dr. Eric Chan-Tin

Dr. Nohpill Park

Name: SREENIVASA SIVA BHANODHAY NANUGONDA

Date of Degree: DECEMBER, 2015

Title of Study: PARALLEL IMPLEMENTATION OF A FORECASTING ALGORITHM

Major Field: COMPUTER SCIENCE

Abstract: This thesis presents the implementation of a time series forecasting method in a High-Performance Computing (HPC) environment. This time series forecasting method integrates concordances and genetic programming (GP). The innovation in this work is the parallelization of a serial program. This is implemented on the OSU Cowboy High-Performance Computing Cluster (HPCC) using Message Passing Interface (MPI) based MPJ Express library. The two components which are parallelized in this thesis are concordance computation of a time series and developing an equation to predict the values of a time series. This parallelization was achieved by distributing the workload of the above-mentioned two components among the available processors for the algorithm to use on HPCC. For being able to parallelize and execute this algorithm on an HPCC, all the steps in the algorithm was executed without any manual intervention unlike in the originally proposed method. To make this algorithm run as a batch the genetic programming part of the algorithm was introduced with a stopping condition which also decided the accuracy of the prediction. So the implementation in this thesis allows the user to choose the accuracy of the predictor function by specifying the mean square error (MSE) limit and using this limit the predictor function development using GP can be stopped. The time taken by the model implemented in this thesis is inversely proportional to the MSE specified by the user.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

1.1 Motivation

A time series is a sequence of numbers recorded at uniform time intervals over a period of time. Analyzing a time series allows one to understand the behavioral statistic and other characteristics of the data in time series and then models can be designed to predict future values based on the previous values in the time series. Among various time series analysis methods, there are some methods which use only comparisons of values and not their actual size.

The fusion model proposed in [1] is one such kind of forecasting model which predicts the future values of a time series based on the behavioral pattern of the previous data. To identify a generic trend, the given input data to this algorithm is analyzed by using those methods which use only comparisons of values and not their actual size. The methods used by this algorithm identify generic trends by computing the different measures of concordances such as Kendall's Tau, Gini's Mean Difference, Spearman's Rho, and a weak interpretation of the Weak concordance. Further, a polynomial function is developed using genetic programming to map this obtained generic trend period (which is hereafter referred as a concordant period) with the values of current or latest time period of the same length as the concordant period. Now this polynomial function is used for predicting the future values of that time series. More details about this model are discussed in section 1.2 and its subsections.

This model proposed in [1] being a good model for forecasting a time series, it can further be upgraded by parallelizing the process of developing the prediction or forecasting function using genetic programming as this takes the most time of the whole process. This model is parallelizable because the polynomial function for forecasting is developed using the conventional genetic programming methodology. In the conventional genetic programming, an initial random population is generated and then for each individual in the population fitness evaluation and genetic operations are performed. In this process except for the cross-over operation both fitness evaluation and mutation operation are independent of the other individuals in the population. Even the cross-over operation can be considered to be independent except that this operation needs another individual to complete. So if the population is made globally available even this operation can be executed parallel for all the individuals in the population. More details about genetic programming are discussed in section 2.2 and its subsections and details about how to parallelize genetic programming is discussed in section 2.3 and its subsection. Other than parallelizing the genetic programming part, there are some more features of this model which can be upgraded and they are discussed in section 1.3.

The parallel model implemented in this thesis improves the performance of the fusion model in terms of time taken by the algorithm for time series forecasting and also it gives users the flexibility of choosing the accuracy of the predictor function by allowing them to specify the mean square error (MSE) limit for the predictor function that is will be developed. The time taken by this model to forecast depends on this feature of allowing the user to specify MSE prior to the execution. The time taken by this algorithm is inversely proportional to the MSE specified by the user.

1.2 Fusion Model

The fusion model proposed in [1] by Mahesh S K, Benjamin P, K.M. George and N. Park is a combination of Concordances and Genetic algorithm to predict the future stock market behavior and values. In this method, the future trends and values of company's stocks are predicted by observing the trends of that company's historic stock prices. Since the historic prices of the company can be very huge, they have come up with an approach where they identify a predictor segment using mathematical concordances to compare with the equivalent recent segment. To calculate the week *Tou, Gini, Kendal's Tau* and *Rho* concordances, all the possible past segments e compared to the current segment. The best matching segment is chosen as the predictor. This resulted in all the lengths and positions for high concordances. Higher the concordances and longer the matches indicated better matches. A high concordance means that the trend is likely to continue, so this high concordant period was used to predict the future trend. For better accuracy in prediction a mathematical equation $g(x)$ was searched for mapping the past data to future data. This search for the mathematical equation was done through genetic programming. This mapping can be represented as,

$$\forall k, g(p_k) \approx f_{k+n}$$

$g\ is\ the\ genetic\ polynomial;$
$p\ is\ the\ past\ data;$
$f\ is\ the\ present\ data;$
$k\ is\ a\ day\ in\ the\ past\ and\ n\ is\ the\ offset\ in\ days;$

They were concentrating on minimizing the mean square error represented by the following function.

$$\sum_{k=1}^{l} (\ g(p_k)\ -\ f_{k+n}\ )^2$$

$$l\ is\ the\ length\ of\ the\ section\ found\ by\ the\ concordance\ measures;$$

The square makes larger differences matter much more than smaller differences. Ideally the genetic polynomial function $g(x)$ will keep getting closer to correct the perfect function after every generation. However in practice an error amount e_k is observed. By extrapolating that error and using the known values from the past, the future values are guessed.

The above-mentioned computations were implemented as a serial process which can be parallelized to make the process yield results at a much faster rate.

1.2.1 Steps Involved in Fusion Model

1. Historical data was downloaded from yahoo finance at http://finance.yahoo.com/market-overview/.

2. From the downloaded historical data file, dates and adjacent closing prices were considered to create a historical dataset. This Historical dataset was flipped to get the latest data at the end of the data set and then the positions of date and data columns were swapped. After all these manipulations, the historical dataset was saved into a new historic file.

3. Using the data in the historic file, the concordances were computed on Matlab tool. Then another file was created with the computed concordances and their corresponding indexes of the data in the historic file created above.

4. Along with the above created two files, the number of days to be predicted was given as arguments to another GUI program which runs GP to generate the predicting equations.

5. This program was started manually and then it keeps generating equations using GP continuously till it was stopped manually again. After this program was stopped the predicted information along with the past information used is saved into a new file.

1.3 Problems in Fusion model

In the fusion model, the genetic polynomial function is developed using conventional genetic programming methodology. The run time complexity of conventional genetic programming makes the design of a standard fusion model algorithm too costly in terms of time.

The concordance computation of segments from the historic data is an independent task in each iteration and hence even this can be parallelized.

The fusion model is designed to be controlled manually to begin and end the process and hence there is specific stopping condition to stop the development of forecasting function using genetic programming. This makes it very uncertain for the end user of this model to decide when to stop the forecasting function to obtain a reliable function to forecast time series.

Other minor features in this fusion model which can be upgraded are,

- The historic data given as input to the fusion model should first be manipulated according to the format recognized by the model. This is an additional effort for the end user of this model.
- In the fusion model, Matlab was used to calculate the concordances of segments from the historic data, which is also an additional effort for the end user of this model.

1.4 Thesis Objective

The objective of this thesis is to address all the problems mentioned in the previous section. As mentioned in that section, the runtime complexity of the Fusion model is high and hence using serial GP is not suitable to analyze huge amounts of data. To overcome this high time consumption, some of the independent part of the fusion model like finding concordances, genetic operations in the process of developing forecasting function can be parallelized.

This parallel fusion model is implemented on a High-Performance Computing Cluster *Cowboy* of Oklahoma State University using Message Passing Interface based MPJ Express library for inter-processor communication.

CHAPTER II

LITERATURE REVIEW

2.1 Parallel Processing

Computing tasks that involve heavy mathematical calculation (highly computationally intensive) or a large amount of data take a long time to complete using only one computer. If such tasks involve independent repetitive computations then it is a known fact parallel processing is much faster than sequential processing. Parallel processing is the processing of program instructions by dividing them among multiple processors with the objective of running program in less time. The earliest example of time taking task is executing a computation intensive program and writing its intermediate result into the external memory tape, so for every IO operation the execution of computation intensive program is idle. So to minimize the total time taken by the whole process an intermediate memory can be introduced to save the information which is to be written to the external memory tape. From then the computer could start an I/O operation and while it was waiting for the operation to complete, it would continue the execution of computation intensive program.

2.1.1 Multiple programs one processor

In the later improvements, multiprogramming was introduced in such a way that a single processor was used to execute multiple programs each for a short time. To the end users, it seemed that all the programs were executing simultaneously but the actual fact was that the programs were being executed concurrently. Logically this is possible with time slicing i.e. dividing available period of time among the processes executing. Though this approach saved considerable amount of time, explicit request for resources would make some processes starve or sometimes create conditions like deadlocks.

2.1.2 Multiple processors

Eventually solving all problems that arrived in parallel processing, the use of multiple processors was introduced i.e. two or more processors were used to complete a single task unlike the case of using a single processor for multiple tasks. In the earliest days of multiprocessor systems, they had master/slave configurations for executing tasks as it was not then understood on how to program a machine so they could cooperate in managing the available resources in the system. One processor that acts as the master processor was programmed to work for the complete task and the rest of the processors that acted as slave processors performed only those tasks assigned by the master processor to them. Later even this problem was solved by using symmetric multiprocessing (SMP) system where each processor was equally capable of managing the workflow in a system and was also responsible for releasing resources timely. Along with this, it was also possible to execute some instructions out of order in some programs which required programmers to deal with increasing complexity in writing programs.

2.1.3 Parallel processing with message passing systems

As the number of processors increased, the time is taken for data accessing increased as it would take more time for the data to propagate from one part of the system to the other part. With this problem existing adding more and more number of processors would never be beneficial in solving a time computation intensive task as the time saved by adding more number of processors was again consumed in data propagation among the processors in the system. So to solve this problem of long propagation times, message passing techniques were introduced in the form a separate system. This message passing system helped programs that shared data by sending messages to each other announcing the change of value to particular operands. Here instead of blindly broadcasting the new value of an operand to the whole system, the new value was passed only to those processors which used that operand. That implies that instead of having a common shared memory a network that supports the transfer of messages between processors can be used. This methodology allows any number of processor to work together efficiently in one system. So the system working with this methodology were known as Massively Parallel Processing (MPP) systems. All those problems using huge amounts of data and which could be broken down into separate individual operations could be executed on MPP systems making it the most successful parallel processing system. Examples of such problems as data mining, where there is a need to perform multiple searches inside a static database and artificial intelligence is another such problem where there is a need to analyze multiple alternatives of possibility to find a solution.

Architecturally most often MPP systems are structured as clusters where within each cluster the processors interact as SMP systems. It is only between the clusters that messages are passed because operands of programs may be accessed via message passing or memory addresses. Programmatically SMP machines are relatively simple while MPP machines are not. SMP machines are more suitable for all type of problems which don’t involve huge amounts of data while for the others MPP is the only suitable system.

2.2 Machine learning

As mentioned earlier machine learning techniques are one such kind of highly computationally intensive task which would definitely need parallel processing systems to perform efficiently. Machine learning is a subfield of artificial intelligence that evolved from the study of pattern recognition and computation learning. Machine learning explores the study and development of algorithms that can learn from data to make predictions in order to construct computer programs for finding solutions. Such kind of algorithms operates based on the inputs and data to make data-driven predictions or decisions rather than following strict static programming instructions. Machine learning is closely related to and most often used for predictive analysis or predictive modeling. Some of the machine learning approach [31, 32] are decision tree learning, artificial neural networks, reinforcement learning, representation learning, genetic algorithms, similarity and metric learning.

Briefing these approaches; Decision tree learning maps observations to possible conclusions from a decision tree which is used as the predictive model. The artificial neural networks are inspired from the structural and functional aspects of biological neural networks. Computations are arranged in the form of an interconnected group of neurons and processing information using connectionism. Neural networks are usually used to draw complex relationship or patterns between input and output data. Reinforcement learning as per the definition is concerned with how an agent is ought to take actions in an environment so as to maximize some notion of long-term reward. Representation learning helps in transforming an input that makes it useful for other machine learning techniques while preserving the actual information in the inputs. Similarity and metric learning help find similarity between pairs provides as inputs based on given pairs of examples which are considered to be similar.

2.3 Time Series Prediction

A time series is a sequence of numerical data points in successive order, usually occurring in uniform intervals. One of the examples of a time series is daily closing stock prices of an organization over a period of time. Incorporating future events into the decision-making process is an important step in forecasting. Time Series Prediction specifically refers to forecasting based on past data in a chronological sequence. Owing to the various factors involved in forecasting events based on past data, and the different forecasting methods with increased accuracies and reduced errors, time series prediction is a challenging task. This study considers an efficient parallel implementation of a genetic programming solution for time-series forecasting.

The problem of forecasting the stock price is quite challenging since the data changes rapidly and is unpredictable. In order to forecast the stock price, and for some financial purposes Genetic Algorithm can be used. Genetic Algorithms (GA) are search algorithms that imitate evolution and natural selection of genetics. The main function of GAs is to generate a population of individuals. The individuals are selected based on the fitness function, it determines how likely individuals are to reproduce. A new population is then evolved by performing the crossover and mutation operations. This process continues until sufficiently fit individuals are generated.

2.4 Genetic Algorithms

Genetic algorithms (GAs) are adaptive heuristic search algorithms based on evolutionary ideas of natural selection and genetics. So GA represents an intelligent manipulation of a random search used to solve optimization problems. GA can be used for solving both constrained and unconstrained optimization problems based on the natural selection process. Although GA generates random generations they are not just random solutions, instead they are based on the historical information provided as a reference to direct the search into better probability within the search space. So at each step, the GA randomly selects individuals from the current population and used them as parents to produce the children for next generation. Gradually over successive generations the population evolves into an optimal solution. Genetic Programming (GP), a branch of Genetic Algorithm (GA), is the origin of computer programs that program themselves. This means that the output of GP is essentially another computer program. Alternatively GP can be described as an evolutionary computation (EC) technique that automatically solves problems without having to tell the computer explicitly how to do it. At the most abstract level, GP is a systematic, domain-independent method for getting computers to automatically solve problems starting from a high-level statement of what needs to be done.

GAs can solve all the optimization problems which can be detailed with the chromosome encoding representation. GAs are uniquely distinguished by having a parallel population-based search with a stochastic selection of many individual solutions, stochastic crossover and mutation [2]. Many other search methods in artificial intelligence and evolutionary algorithms have only some of these features whereas only GAs has all these features combined together. Applications of GA can be found in various fields such as engineering, biotechnology, economics, manufacturing, pharmacology, chemistry, biology, mathematics, computer science and medicine. GA has a statistical function to evaluate the fitness of an individual in generations and hence this algorithm is used for searching solutions of high complexity which more often involves large non-linear discrete data as input. GA are preferable over many traditional methods as they consider variables as string codes instead of simple variables which mean the scope of searching is discretized though GAs function might have been continuous. Another accountable advantage of GAs is that the searching of the solution is based on the resemblance of string structures which makes comparisons in searching easy. Some more notable advantages of GA is the use of probabilistic approach for finding solutions instead of deterministic rules, they work with program code for solution instead of relying on solution itself, they work with population of solutions instead of one solution, they use population-based search which will decrease the amount of work to be done by the user in applying the same algorithm multiple times as that is already done by the GA on the complete population. This way GA have many advantages over traditional methods which can be used to increase the probability of finding the optimal solution.

2.5 Genetic Programming

Genetic Programming (GP) is an evolutionary algorithm where individuals in the population are computer programs. So in each generation GP transforms the population of computer programs into another program with varied features from the other members in the population [3].

GA have many advantages over traditional methods which can be used to increase the probability of finding the optimal solution.

2.5.1 Algorithm of GP

1. Randomly create an initial population of mathematical functions from the available primitives.

   Repeat

2. Compute each function and ascertain its fitness. Sort the population based on fitness value.

3. Select one or two functions from the population with a probability based on fitness to participate in genetic operations.

4. Create new individual functions by applying genetic operations using the above-selected functions.

   Until

5. A function is generated which gives a mean square error less the given value.

6. Return the best-so-far individual i.e. that function which met the above condition.

The two main operations of GP known as genetic operations are,

- **Crossover**: A new individual is created by combining randomly chosen parts from previously generated individuals.

- **Mutation**: A new individual is created by randomly manipulating a previously generated individual.

2.5.2 Representation of individuals in GP

Programs (individuals) in GP are represented in the form of *syntax trees* instead of lines of codes. For example consider a mathematical function or equation *2 + 3 + (X*7) + (Y/5).* The equivalent syntax tree is *(+ 2 3 (* X 7) (/ Y 5)).* Graphical representation of this syntax tree is

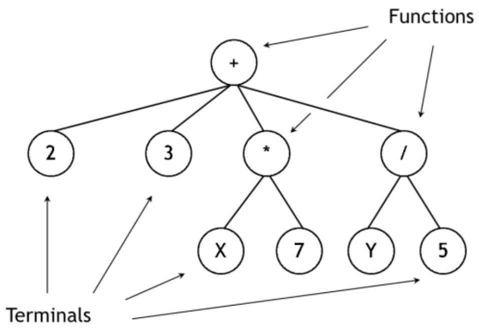


Figure-1: Graphical representation of syntax tree [27]

The definition of terminal set and function set to specify a language for the GP to create programs.

Function set: The function set is usually driven by the nature of the problem domain. Function set for the syntax tree given in previous page is {+, -, *, /}

Terminal set: The terminal set mostly consists of the program's external inputs such as variables. In some implementations, the terminal set may also include functions with no arguments which yield constants as results. Terminal set for the syntax tree given in the previous page is {integers, X, Y}

The set of allowed functions and terminals together from the *primitive set* of a GP system.

2.5.3 Genetic Operations

Initialization of population: a set of the population is randomly generated from the available primitive set. The member of the population is generated by randomly selecting either a function or a terminal or both to represent the program. If a function was selected, recursively random programs should be generated to act as arguments.

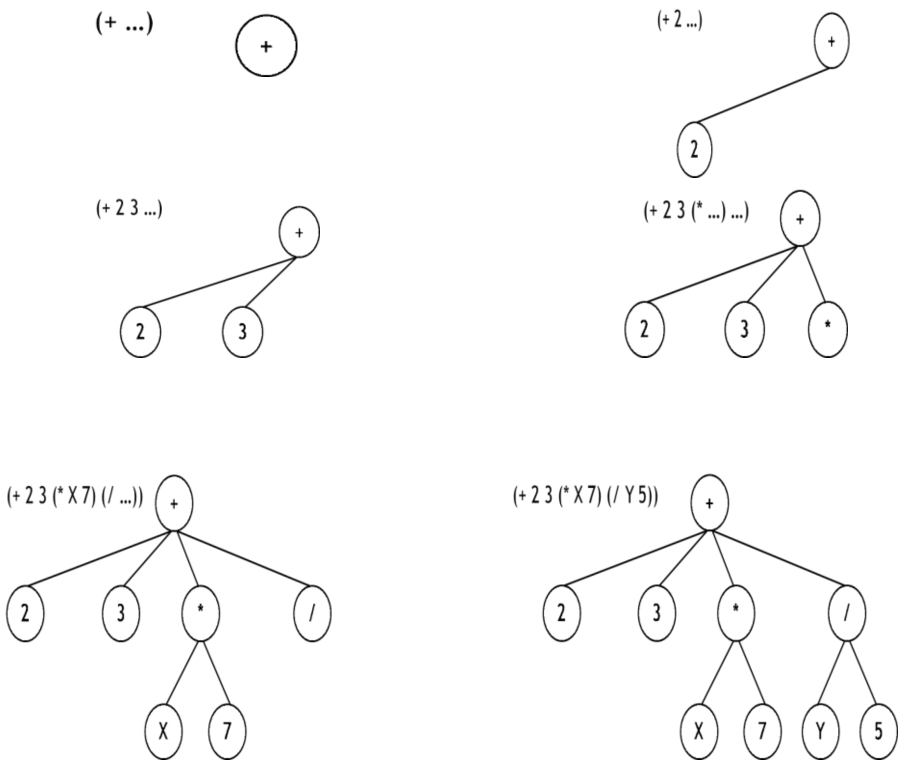


Figure-2: Population Initialization [27]

Selection: Genetic operations in GP are applied on those individuals that are selected probabilistically based on their fitness. So, individuals with better fitness are most probably involved in genetic operations to produce child programs in future generations. The most usual method of selection used in GP is tournament selection, followed by fitness proportionate selection though any standard evolutionary algorithm selection mechanism can be used.

Crossover or Recombination: Given two parents/individuals, subtree randomly selects a crossover point in each parent tree. Then, it creates the offspring by replacing the sub-tree rooted at the crossover point in a copy of the first parent with a copy of the sub-tree rooted at the crossover point in the second patent as shown below.

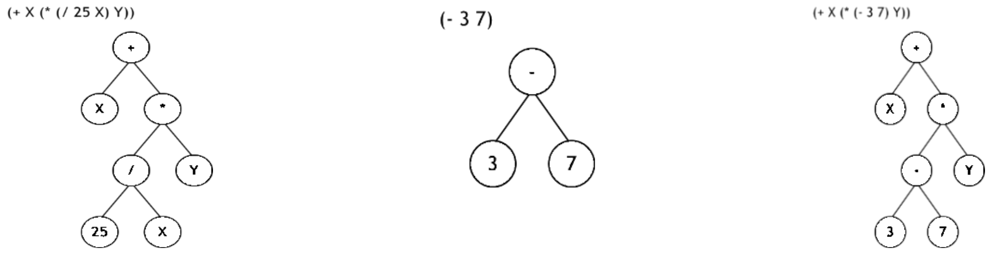


Figure-3: Crossover [27]

Mutation: Mutation in GP randomly selects a mutation point in a tree and substitutes the sub-tree rooted there with a randomly generated sub-tree. Another form of mutation is a point mutation, in which a random node is selected and the primitive stored there is replaced with a different random primitive of the same arity taken from the primitive set. If no other primitive with that arity exist, nothing happens to that node (but other nodes may still be mutated).

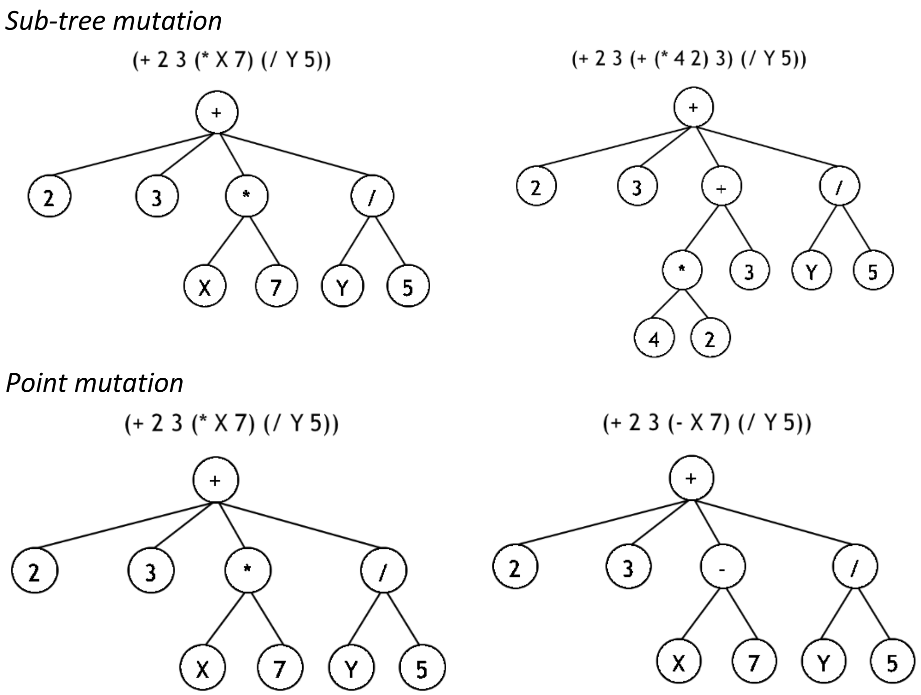


Figure-4: Mutation [27]

Fitness measure: This is that value which decides the survival of a population. The search scope of the GP for the desired individual is the initial primitive set defined by the combination of terminal set and function set. This includes all the programs that can be constructed by composing the primitives in all possible ways. From this scope of the region, it is the task of the fitness measure to find the individual that produces the desired result i.e. the fitness measure is the sole mechanism that defines a high-level statement of the problem’s requirements to the GP systems.

2.6 Parallel Genetic Programming (PGP)

Genetic programming uses parse trees instead of code syntaxes that represent executable programs solving problems. These parse trees consist of nodes as in a simple graph with leaf nodes at the bottom. To serve the purpose of generating functions or programs for solving complex problems the standard GP can go beyond the production of parse trees. However, the run-time complexity of GP makes the design of a standard algorithm and its implementation too costly in terms of time. To overcome the high computation cost of conventional GP techniques, Parallel Genetic Programming (PGP) has been developed. Genetic programming can be parallelized by introducing multiple communicating populations same as the natural evolution of spatially distributed populations [12]. Similarly, there are many other methods available to parallelize genetic programming which will be discussed in next subsection. M. Oussaidene states in [10] that most complex applications, such as trading model optimization, the evaluation of each program (individual) is very long and thus the time spent in the selection and reproduction phases is practically negligible compared with the population evaluation time. Fernandez and other authors in [11] studied the parameters that affect parallel performance and study their interactions in common problems to develop a more robust model of how parameters might be set so as to maximize the performance of parallel GP in many different types of problems. PGP is a computationally intensive technique which is also highly parallel in nature and so there have been significant performance improvements over a standard GP in single-processor based approach by harnessing the parallel computational power of many-core graphic cards (Modern PC graphics cards contain powerful GPUs including a large number of computing components) which have hundreds of processing cores [3]. This enables both fitness and individual solutions to be evaluated in parallel. Despite the increased use of PGP for better results, population size and the choice of an appropriate PGP model were considered as challenges for the parallel approach [13, 14]. Previously a number of PGP methodologies were studied for optimizing the results, considering the challenges mentioned earlier. Some of the PGP models are the Master-Slave model, the Coarse-Grain model, and the Fine Grain model [12, 15]. In the Master-Slave model, the population is stored in the master node and a fraction of the population is assigned to each available slave processor and the slave evaluates and returns the fitness value [16]. In the Coarse-Grain model, the population of individuals is divided into several autonomous subpopulations, called Demes, which exchange individuals at a certain rate, called the migration rate. Fernandez and Tomassini studied the relationship between the classical model and the island models of the coarse-grain model of PGP [11]. On the other hand, in the Fine Grain model, the population is divided into a large number of small subpopulations which are assigned to independent processes. This method is suitable for machines consisting of multiple processors (massive parallel architectures).

2.6.1 Parallel Genetic Programming Models

The evaluation of fitness, mutation, type of subpopulations and the selection process decides the way of parallelizing the genetic algorithm. Nowostawski classified parallel GA/GP into eight categories namely, Master-Slave parallelization, Static Subpopulations with Migration, Static Overlapping Subpopulations without Migration, Massively Parallel Genetic Algorithms, Dynamic demes, Parallel steady-state genetic algorithms, Parallel messy genetic algorithms and Hybrid methods [19].

The Master-Slave GA uses a single population, but the evaluation of fitness is distributed among several processors. The selection and crossover consider the entire population. In this GA, the master runs the genetic algorithm, controls the slaves and distributes the population among the slaves whereas the slaves take the population from the master, evaluate their fitness and send their fitness back to the master [17].

The static subpopulations with migration parallel GA make use of multiple demes and the migration operator. These algorithms are also known as coarse-grained GA or multi-deme GA. This is the most often used model of parallel GA. In this model, the population is divided into several subpopulations called demes. Multi-deme GA consists of several subpopulations which exchange individuals occasionally. This exchange of individuals is called migration. Multi-deme parallel GA is also known as distributed GAs as they are implemented on distributed memory MIMD computers [18].

The Overlapping Subpopulations without Migration model is similar to the previous one except for the lack of migration operator. Instead, propagation and exchange are done in individuals which lie in the so-called overlapping areas.

Massive Parallel Genetic Algorithms have a single population with spatial structures limiting the interaction between individuals. This model is suited for massively parallel computers, but it can be implemented on any multiprocessor.

Dynamic Demes is a new parallelization technique which combines global parallelism with coarse-grained GA. In this model, there is no migration operator as such, because while the population is treated during evolution as a single collection of individuals, and information between individuals is exchanged via dynamic reorganization of the demes during the processing cycles. This model is scalable and easy to implement.

The Parallel Steady-State Algorithms is completely straightforward to parallelize the genetic algorithm operators since this kind of GA uses continuous population update schemes. If children are gradually introduced into a single, continuously-evolving population, the only thing to do is apply selection and replacement schemes in the critical section. The other GA operators, including fitness evaluation, can be run in parallel.

As stated in [19] the way in which GPs can be parallelized depends on the following elements:

- How fitness is evaluated and mutation is applied.
- If single or multiple subpopulations (demes) are used.
- If multiple populations are used, how individuals are exchanged.
- How selection is applied (globally or locally).

Depending on how each of these elements is implemented all the above mentioned parallel methods were classified. This thesis parallelizes the fusion model for time series forecasting based on Genetic Programming. In GP of this model, initially 1024 equations are generated which is a single initial population. The fitness evaluation, mutation and crossover application should be done for each equation in order to develop the desired equation. This is done by dividing the total population into multiple subpopulations (demes). Selection is applied globally by considering all the equations after performing genetic operations. This model can be viewed as a combination of Master-Slave parallel model and the Static subpopulations with migration. There is a master-slave system which controls the distribution of tasks (genetic operations). After these genetic operations are performed on the individuals, their fitness calculation is also done in parallel in the slave nodes. After all these operations are completed, all the distributed population are gathered (migrated back) into the master node for checking the stopping condition. The advantage of this model is that it avoids the communication between subpopulation which might take more time instead this model migrates all the developed equations to the master node to be used for generating the next generation. Other than these benefits, the model implemented in this thesis parallelizes the serial process proposed in the previous Fusion model proposed in [1].

2.7 Prediction using Parallel Genetic Programming

Owing to the challenges in the forecasting financial market many approaches have been employed such as portfolio optimization, bankruptcy prediction, financial forecasting, fraud detection, and scheduling. These techniques used recurrent neural networks combined with PGP for the daily stock trading [20].

M. Oussaidene and B. Chopard presented a scalable parallel implementation of GP on distributed memory machines [21]. The system described runs multiple master-slave instances interacting asynchronously. Master-Slave Parallelization is a significant approach which can further be classified as Asynchronous and Synchronous. Typically, the fitness function is parallelized. Master stores the population or a fraction of the population is assigned to each available processor and slave evaluates and returns the fitness value [16]. Synchronous master-slave PGA is relatively an easier mechanism as the master waits for the slave to return all the fitness values before proceeding to the next generation. Though it has the same properties as that of a simple GA, enhanced speed gives it a high edge. However, since the whole process has to wait till the slowest of the processors completes its fitness evaluations, asynchronous master-slave PGA is taken as an alternative. Considering only a fraction of individuals in the population whose fitness has already been evaluated, makes it better and easier to implement [10, 15].

Munawar states that the evolving parallel paradigms help researchers working on GA/GP by making it easy for them to parallelize GA/GP and at the same time this parallelization task have added a research area for the High-Performance Computing (HPC) community [22]. Though GA has a high potential for parallelization, it was common practice to implement GAs in a serial fashion. But this resulted in rising clock speeds which mean a rise in power consumption. Hence, the multicore strategies helped achieve high performance with low power consumption taking PGP to the next level. Distributed Computing, allows the use of resources distributed at global distances. All these changes made parallel and distributed algorithms more vital than before [22].

2.8 Kendall Rank Correlation Coefficient or Kendall's Tau (τ)

In fields like medical, economy, astronomy etc... historical information is in the form of time series. With the development of monitoring equipment, these kinds of data will become abundant and hence there is a need for methods of time series analysis which apply directly on real data by only comparisons of values and not their actual size.

τ also known as Kendall's tau coefficient, developed by Maurice Kendall [4] is a statistical measure used to measure the association between two measured quantities. It is known as a good measure for determining the similarity of movements of time series. Let $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$be two sets of size n, where (x_1, y_1), (x_2, y_2) (x_3, y_3)…. (x_n, y_n) are the set of observations of the random variables X and Y respectively. A pair of observations (x_i, y_i) and (x_j, y_j) are said to be concordant if $(x_i - x_j) * (y_i - y_j) > 0$. The Kendall's tau (τ) is defined by Equation 1, where n_c is the number of concordant pairs, n_d is the number of discordant pairs, and n is the number of elements in each dataset. The denominator of the equation is the total number of combinations of different pairs, so the range of the coefficient is the interval [-1, 1].

$$\tau = \frac{n_c - n_d}{\frac{1}{2}n(n-1)} \qquad (1)$$

Where $n_c$number of is concordant pairs and n_d is number of discordant pairs.

An alternate form of Kendall’s Tau is shown below,

$$\tau = \frac{2\left[\sum_{i=1}^{n-1}\left(\sum_{j=i}^{n} \operatorname{sign}(Xi - Xj) * \operatorname{sign}(Yi - Yj)\right)\right]}{n(n-1)} \qquad (2)$$

Series 1: X1, X2, X3…... Xn; Series 2: Y1, Y2, Y3…... Yn;

- If the agreement between the two pairs is perfect (i.e., the two pairs are the same) the coefficient has value 1.
- If the disagreement between the two pairs is perfect (i.e., one pair is the reverse of the other) the coefficient has value −1.
- If X and Y are independent, then we would expect the coefficient to between -1 < τ > 1.

2.9 High-Performance Computing Cluster (HPCC)

High-performance computing is a practice of using multiple computing elements to solve a larger problem in science, engineering or business to achieve higher performance than one could get using a regular computer. Usually, a high-performance computer today is typically a cluster of simple computers commonly used by anyone. They all have processors, memory, disk, operating systems and other peripherals but are more in number. Each computer in a cluster is referred to as a node by HPC users. For these clusters to work towards solving a common problem, they need to communicate with each other which is done over networks and there is a variety of interconnecting networks available such as Ethernet, InfiBand etc. So a high-performance computer is usually built from what are many basic or ordinary computers connected together with a network and centrally coordinated by some special software. Since this computer is physically very close together, the cluster is a common term for a high-performance computer.

Oklahoma State University has a High-Performance Computing Cluster COWBOY (figure-5) [5], which was funded by NSF MRI grant “Acquisition of a High-Performance Compute Cluster for Multidisciplinary Research”. This cluster Cowboy from Advanced Clustering Technologies consists of following features:

- 252 standard compute nodes, each with dual Intel Xeon E5-2620 “Sandy Bridge” hex core 2.0 GHz CPUs and 32 GB of 1333 MHz RAM.
- Two “fat nodes” each with 256 GB RAM and an NVIDIA Tesla C2075 card.
- A 92 TB of globally accessible high-performance disk provided by three shelves of Panasas ActivStor12, this includes 20 2TB drives and the aggregate speed of the disk is 4.5GB/s read and 4.8GB/s write.
- Interconnect networks are InfiBand for message passing, Gigabit Ethernet for I/O and an Ethernet management network.

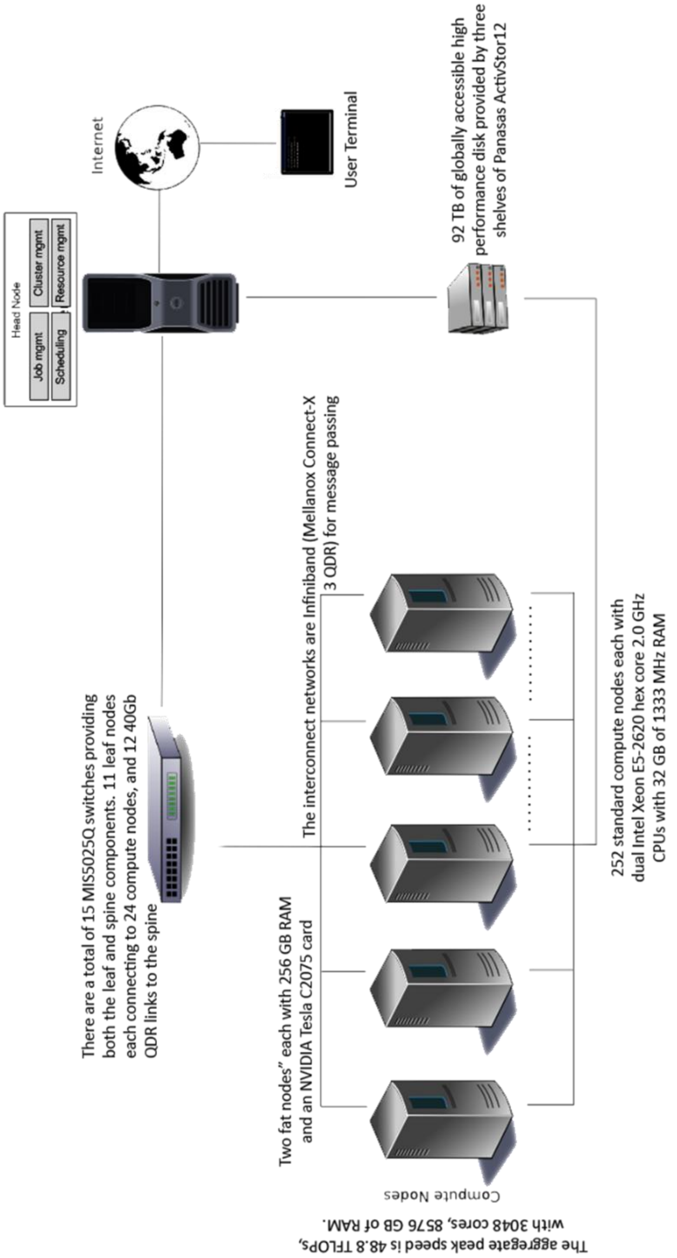

Head Node
Job mgmt
Cluster mgmt
Scheduling
Resource mgmt
Internet
User Terminal
92 TB of globally accessible high performance disk provided by three shelves of Panasas ActivStor12
There are a total of 15 MIS5025Q switches providing both the leaf and spine components. 11 leaf nodes each connecting to 24 compute nodes, and 12 40Gb QDR links to the spine
The interconnect networks are Infiniband (Mellanox Connect-X 3 QDR) for message passing
Two fat nodes" each with 256 GB RAM and an NVIDIA Tesla C2075 card
252 standard compute nodes each with dual Intel Xeon E5-2620 hex core 2.0 GHz CPUs with 32 GB of 1333 MHz RAM
Compute Nodes
The aggregate peak speed is 48.8 TFLOPs, with 3048 cores, 8576 GB of RAM.

Figure-5: OSU HPCC Cowboy from Advanced Clustering Technologies [28]

2.10 Message Passing Interface (MPI) [26]

As mentioned earlier in section 2.1.3, parallel processing with message passing systems has emerged as an efficient model of parallel processing systems for parallel programming. Message Passing Interface (MPI) is a specification for a standard Library for message passing that was defined by the MPI Forum, a broadly based group of parallel computer vendors, library writers and application specialists. MPI is a message-passing application program interface equipped with protocols and semantics that decides its behavior in any implementation of parallel programming on a message passing system. MPI features both point-to-point and collective message passing operations. MPI provides abstractions for processes at two levels. First, processors are named according to the rank of the group in which the communication is performed. Second, virtual topologies allow for a graph or Cartesian naming of processes that help relate the application semantics to the message passing semantics in a convenient, efficient way. Communicators in MPI provide a significant measure of safety useful for building library-oriented parallel code [6]. MPI is well defined with constructors and destructors which make its functionality be an opaque object-based program. These objects include Groups (the fundamental container for a process), Communicators (which contain groups used as arguments for communication calls), and request objects for asynchronous operations. MPI also features user-defined and predefined datatypes allowing for heterogeneous communication and sophisticated description of gather/scatter semantics in send/receive operations in both point-to-point and collective operations. MPI supports both Single Program Multiple Data and Multiple Program Multiple Data models of parallel programming. MPI provides a thread safe application programming interface (API), which will be of use in the implementation of multithreaded program environments and also support thread safety for themselves.

2.11 MPJ Express [24, 25]

Java is one of the few mainstream programming languages and is multi-threaded by design, it makes an attractive language for programming SMP and multicore clusters, provided a thread-safe communication library is available. MPJ Express is a thread-safe Java messaging library, a quality implementation of MPI-like bindings for Java compatible with Java threads [7, 8]. As the current trend of parallel programming is based on Symmetric Multi-processor (SMP) [section 2.1.2] and multicore clusters a thread-safe HPC library is very important. MPJ Express is a thread-safe communication library alternative to traditional approaches like hybrid MPI, OpenMP, shared memory devices code in the MPI libraries. Other significant advantages of this library are that,

- It provides execution of parallel programs in two configurations [9] with the same code for both the configurations,
    - Multicore Configuration: This configuration allows users to write and execute parallel java applications on their desktops or laptops i.e typically hardware which contains shared memory and multicore processors. In simple words, the MPJ Express initiates a single thread to represent an MPI process, using efficient inter-thread communication mechanism. This allows the users to first develop the code on their personal laptop/desktops with help

    - Cluster Configuration: This configuration allows users to execute their parallel java applications using on distributed memory platforms including clusters or network of computers.

- It provides a debugger for inspecting the execution of a parallel program.

The latest version of MPJ Express available today is version 1.4 which features the following thread-safe communication devices:

  - niodev based on Java NIO used to execute programs on clusters using Ethernet.
  - mxdev based on the Mytrinet eXpress(MX) library used to execute programs on cluster connected by Mytrinet express interconnects.
  - hybdev used to execute programs on clusters of multicore computers.
  - native used to execute programs on top of native MPI Library such as MPICH, OpenMP or MS-MPI.

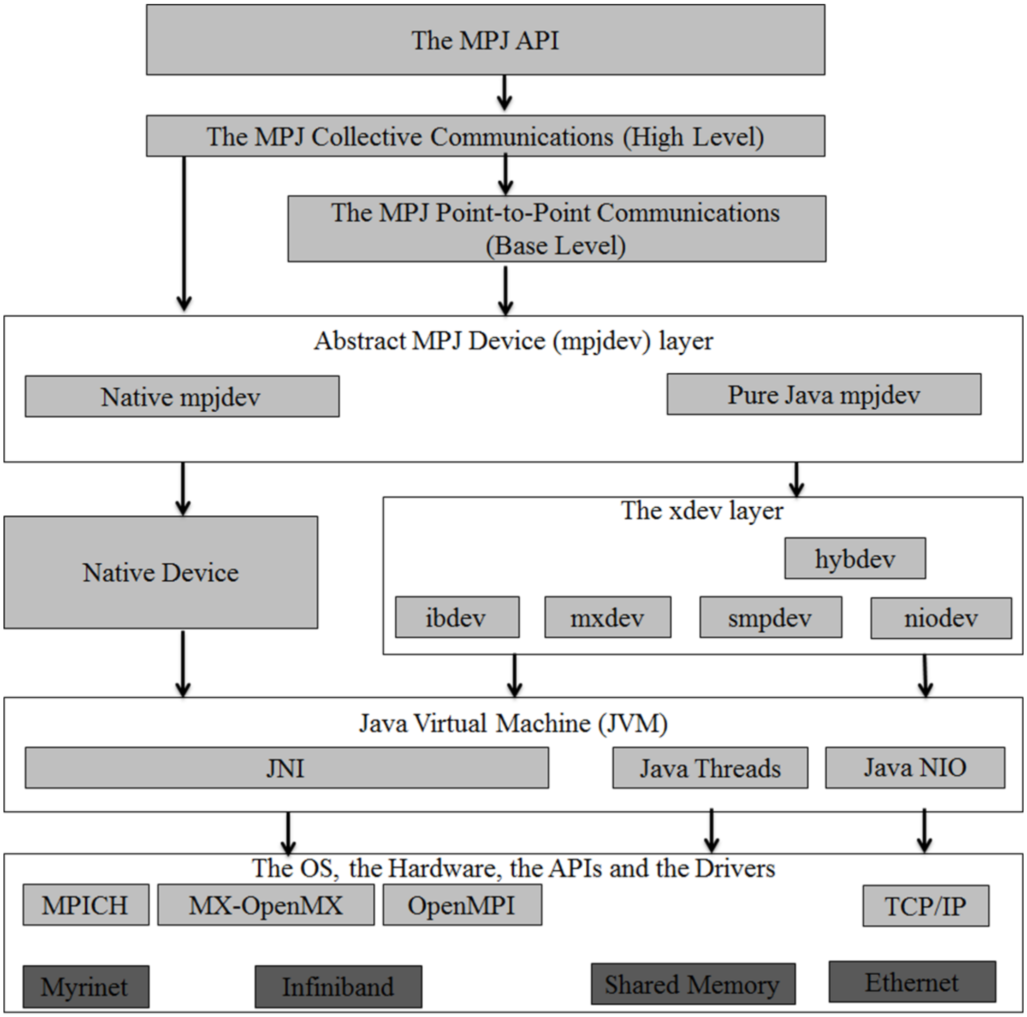

The MPJ API
The MPJ Collective Communications (High Level)
The MPJ Point-to-Point Communications (Base Level)
Abstract MPJ Device (mpjdev) layer
Native mpjdev
Pure Java mpjdev
Native Device
The xdev layer
hybdev
ibdev
mxdev
smpdev
niodev
Java Virtual Machine (JVM)
JNI
Java Threads
Java NIO
The OS, the Hardware, the APIs and the Drivers
MPICH
MX-OpenMX
OpenMPI
TCP/IP
Myrinet
Infiniband
Shared Memory
Ethernet

Figure-6: MPJ Express Architecture [29]

2.12 MPI Program Structure

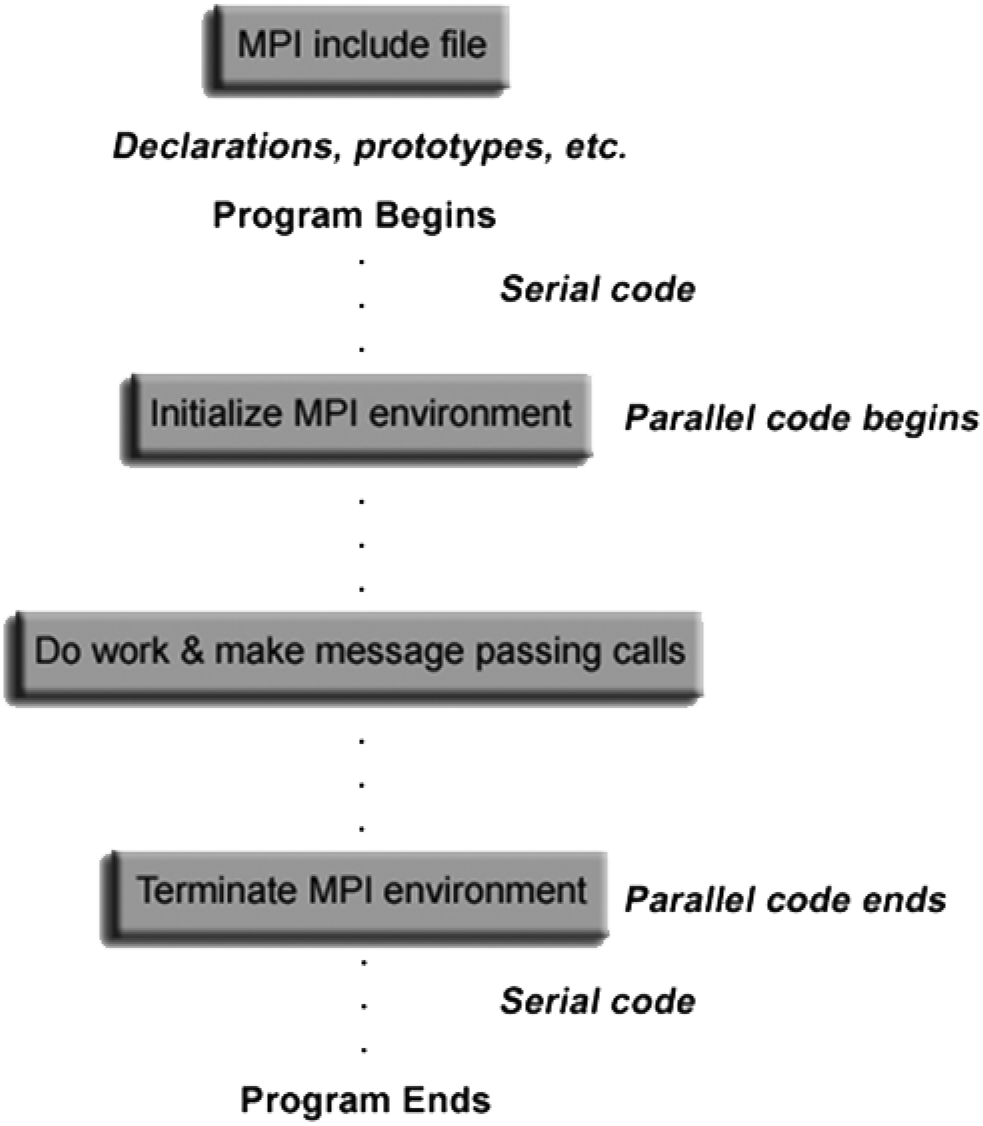

MPI include file
Declarations, prototypes, etc.
Program Begins
Serial code
Initialize MPI environment
Parallel code begins
Do work & make message passing calls
Terminate MPI environment
Parallel code ends
Serial code
Program Ends

Figure-7: MPI Program Structure [30]

CHAPTER III

IMPLEMENTATION

3.1 Overview

This thesis implements parallelization of Fusion Model proposed in [1]. The forecasting method implemented in this research views the forecast as a function

$$f : X \rightarrow Y$$

Where X and Y are sequences or segments of time series. The method involves three parts,

1. Identification of Y
2. Identification of X
3. Construction of f

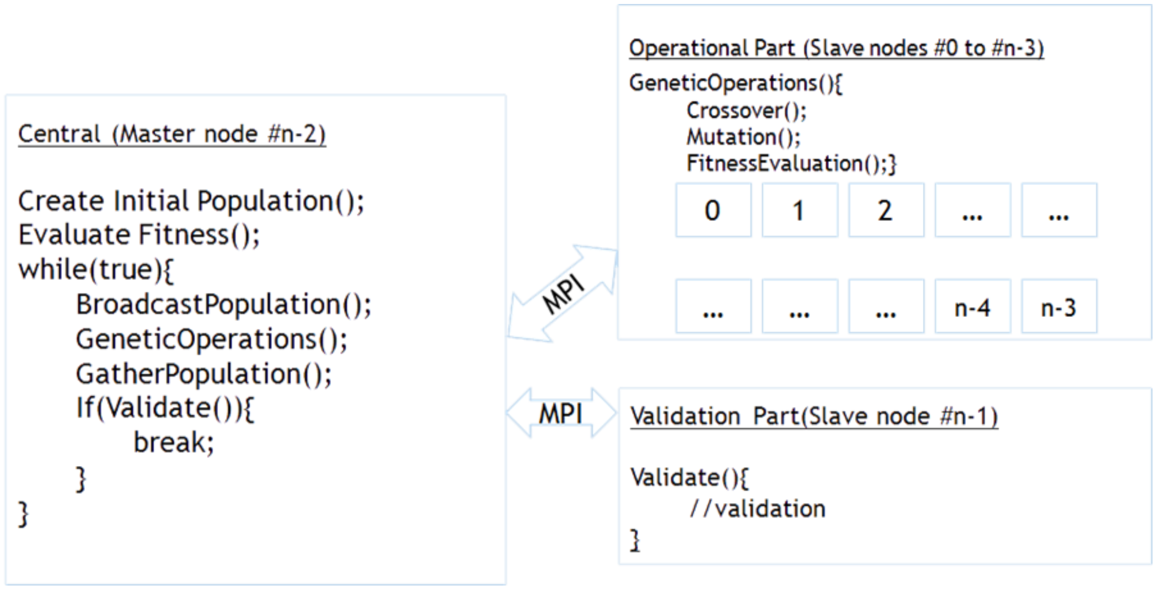
Y is usually the most recent segment of the series of length n. To determine X (the predictor), we use Kendal's τ. Finally, to build f, we use genetic programming. This research implements a program in high-performance computing cluster (HPCC).

3.2 Design of implemented model

The implemented model in this thesis consists of three major parts, they are

- Central:This part of the model controls begin and end of the whole process.
- Operational: This part of the model executes the assigned tasks by the central part.
- Validation: This part of the model validates each generation of the genetic programming.

These three parts of the model communicate with each other using Message Passing Interface. This model is designed in such a way that the last node of the allocated nodes acts as Validation part, the penultimate node acts as Central part and rest of the allocated nodes act as Operational part of the model.


Central (Master node #n-2)
Operational Part (Slave nodes #0 to #n-3)
0
1
2
...
...
...
...
...
n-4
n-3
MPI
MPI
Validation Part(Slave node #n-1)

```
Create Initial Population();
Evaluate Fitness();
while(true){
    BroadcastPopulation();
    GeneticOperations();
    GatherPopulation();
    If(Validate()){
        break;
    }
}
```

```
GeneticOperations(){
    Crossover();
    Mutation();
    FitnessEvaluation();}
```

```
Validate(){
    //validation
}
```

Figure-8: Implemented Model

3.3 Upgrades to the Fusion Model

In this thesis, there have been made few upgrades to the Fusion Model along with the parallelization of Fusion Model. Following are the list of upgrades made to the Fusion Model,

- This thesis implements batch processing of the fusion model i.e. the execution of all tasks in the previous model without manual intervention.
- In this thesis, the historic file downloaded from the Yahoo finance can be given to this model directly without any external manipulation.
- This thesis provides the flexibility to users by allowing them to specify the following information in every run,
    - The size of segments to be created in the process of concordance computation.
    - The length of difference to create segments in the process of concordance computation.
    - Prediction length i.e. number of days from the latest date to predict values.
    - Sum of mean square difference: This value is used as a stopping condition explained below.
- The fusion model previous proposed does not have a stopping condition for the GP.
    - In this thesis, a stopping condition is added to GP by calculating the sum of mean square error (MSE).
    - In every generation of GP, the equation with the best fitness is chosen to compute the sum of mean square error and this value is compared with the MSE specified by the user. If obtained is less than or equal to the MSE specified by the user then the GP stops.
- In the previous model, if the process of concordance computation and predictor equation generation is divided into tasks then we can observe that there are many tasks independent of another task.

- In this thesis, all the independent task are parallelized. The task assignment depends on a number of processors available.

3.4 Algorithm of Implementation

1. Get historical stock price data of a company.
2. Divide this history stock data into chunks and search for a pattern in the past that look similar to the present pattern using concordance measures.
3. Find *Y* the most recent segment of the series of the specified length n.
4. Find *X* the highest concordant past pattern among all the patterns found.
5. Develop a forecasting function *f* using the genetic algorithm for prediction of future data using the lastly found highest concordant past pattern.

3.5 Finding the Predictor

Since this work depends on a huge set of historic prices, a part of historic prices are considered to compare with the present data for prediction. , Concordances are calculated for all possible small sets of data and the set of past data with highest concordant value is chosen for prediction purpose. Kendall's Tau (τ) is used to calculate the above said concordances. As mentioned above Kendall's Tau is a statistical measure that can be used to determine the relation between two sets of quantitative data. In this model, to achieve this the huge set of historic data is divided into maximum possible chunks based on available processors for calculating the concordances for any two sets and all these concordances and their respective data indexes are stored in order to consider them for future purposes.

3.6 Developing the forecasting function

As this parallelization is implemented on a distributed computing environment, the GP can store the intermediate data in distributed memories and these data can be passed on among the nodes of this model using MPI. The parallel GP implementation has three parts namely central, operational and validation. This system is as showed in the following figure,

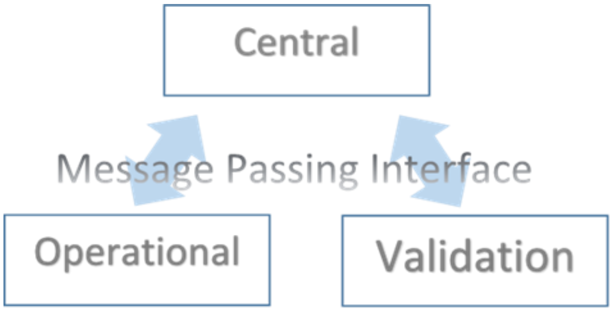


Figure-9: Three Parts of the Implemented Model

3.6.1 Functionality of each part in the implemented model

The central part controls the start and stop of the GP by communicating with the operational part for intermediate operations and also with the validation part for checking the stopping condition of the GP, the operational part performs the genetic operations by receiving population from the central part. The validation part receives the set of populations from the central part after a certain number of GP iterations to evaluate the fitness and returns the status of GP to either continue or stop the process. When a stop signal is sent, then a resulting fit equation is also sent to the central node for outputting the prediction of time series form that fit equation. In this parallel GP system, each part of the system is internally an individual node (central and validation parts) or a set of nodes (operational part) independent from other parts of the system. The functionality of each part is described below,

Central Part

- *Concordance Computation*
  - The central part of the system divides the historic data into segments. The latest segment is considered as *Y*.
  - Rest of the segments are broadcasted to operational part for concordance computation. After the computation, the computed concordances are sent to the validation node.
- *Genetic Programming to find function*
  - The central part controls the start and stop of the GP by communicating with the operational and validation parts

Operational Part

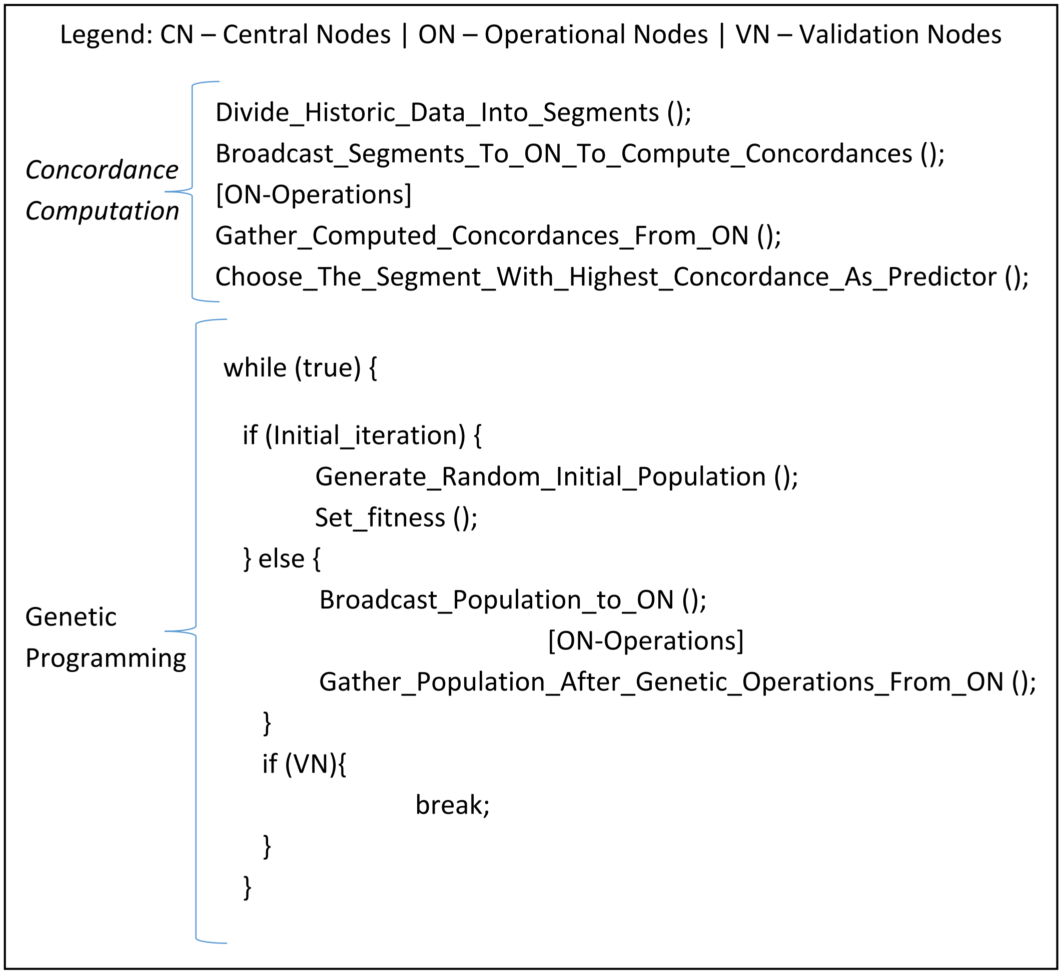
- *Concordance Computation*
  - Kendall's τ is calculated for each segment simultaneously.
- *Genetic Programming to find function*
  - Genetic operations are performed on individuals distributed by the central part.

Validation Part

- *Concordance Computation*
  - From all the computed concordances, the segment with the highest concordance is considered as *X*.

- *Genetic Programming to find function*

    - The stopping condition for GP is checked here. If any equation meets the stopping condition, then future prices are predicted and saved into a CSV file. The GP is terminated at this point.



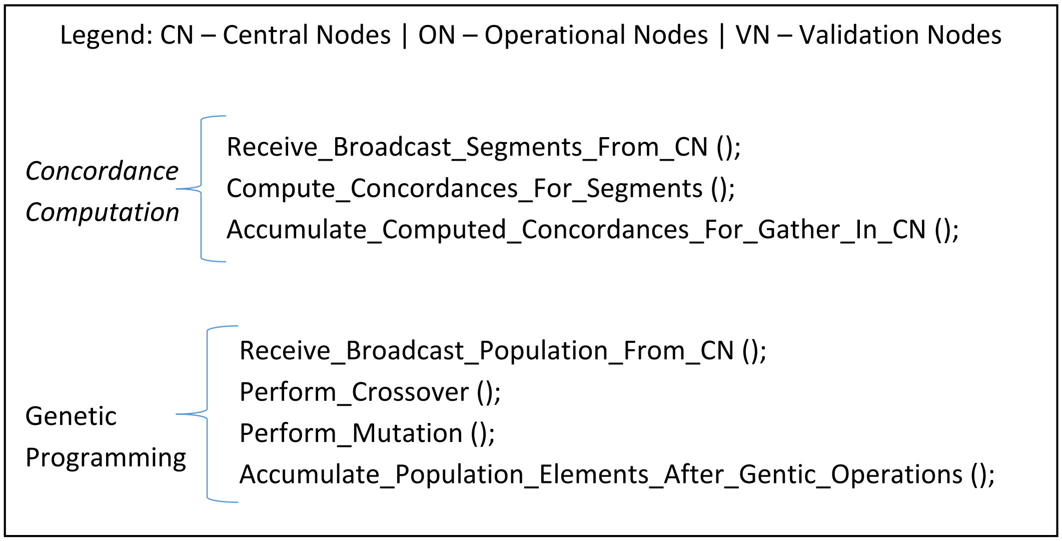
Legend: CN – Central Nodes | ON – Operational Nodes | VN – Validation Nodes

```
Divide_Historic_Data_Into_Segments ();
Broadcast_Segments_To_ON_To_Compute_Concordances ();
[ON-Operations]
Gather_Computed_Concordances_From_ON ();
Choose_The_Segment_With_Highest_Concordance_As_Predictor ();

 while (true) {

   if (Initial_iteration) {
             Generate_Random_Initial_Population ();
             Set_fitness ();
   } else {
              Broadcast_Population_to_ON ();
                                    [ON-Operations]
              Gather_Population_After_Genetic_Operations_From_ON ();
     }
     if (VN){
                        break;
     }
   }
```

Figure-10: Central Part Functionalities



Legend: CN – Central Nodes | ON – Operational Nodes | VN – Validation Nodes

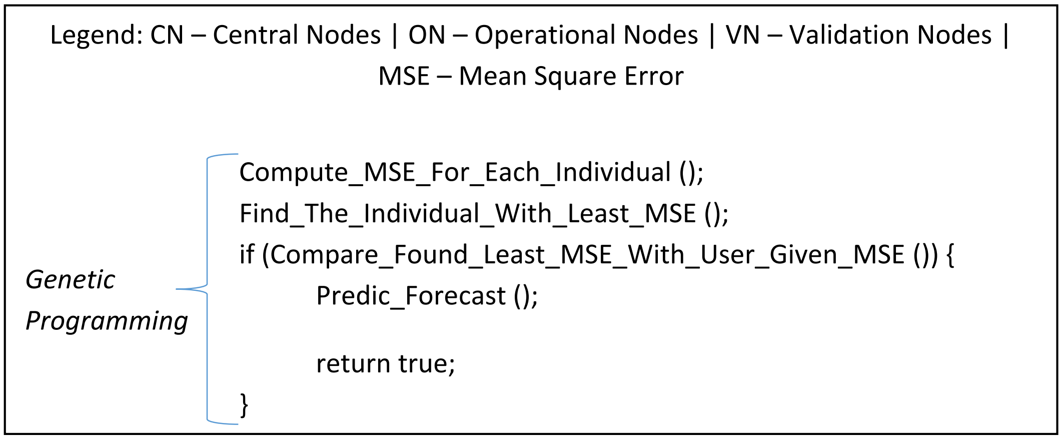
Concordance Computation

```
Receive_Broadcast_Segments_From_CN ();
Compute_Concordances_For_Segments ();
Accumulate_Computed_Concordances_For_Gather_In_CN ();
```

Genetic Programming

```
Receive_Broadcast_Population_From_CN ();
Perform_Crossover ();
Perform_Mutation ();
Accumulate_Population_Elements_After_Gentic_Operations ();
```

Figure-11: Operational Part Functionalities



Legend: CN – Central Nodes | ON – Operational Nodes | VN – Validation Nodes | MSE – Mean Square Error

Genetic Programming

```
Compute_MSE_For_Each_Individual ();
Find_The_Individual_With_Least_MSE ();
if (Compare_Found_Least_MSE_With_User_Given_MSE ()) {
        Predic_Forecast ();

        return true;
}
```

Figure-12: Validation Part Functionalities

3.6.2 Stopping condition for validating the Forecasting Function

The validation part of the implemented model checks if the generated functions satisfy the stopping condition in order to stop the further generation of GP. So this section explains what this stopping condition is and how is it verified and what would happen if this condition is ignored and continued further.

The forecasting method implemented in this thesis views the forecast as a function,

$$f: X \to Y$$

Where f is the forecasting function that is being developed using genetic programming, X is the concordant period and Y is the predicted period of data. So using this function the future values are extrapolated based on the values after the concordant period. The Mean Square Error (MSE) [33] can be used to assess the quality of the predictor function or the forecasting function. As the value of MSE decreases that implies that the accuracy of the forecasting function might increase. The mean square error of a predictor measures the average of the square of the errors where the error is the difference between the actual and the predicted values. Therefore, the MSE can be calculated as follows,

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(f(X) - \hat{\mathrm{Y}}\right)^2$$

Where n is the length of X and Ŷ is the actual data in the base period. In this implementation, a best individual is selected from the population based on fitness in every generation. MSE for this individual is calculated and compared with the MSE limit (value) specified by the user at the start of the algorithm. If this obtained MSE is less than MSE limit specified by the user, then the GP is stopped from developing further generations and the best individual selected will be considered as the forecasting function or the predictor function. Therefore, this MSE limit specified by the user is the stopping condition of the implemented algorithm.

So, based on the MSE limit specified by the user, following two output factors are dependent,

- Accuracy: As the MSE limit (value) decreases the accuracy of the predictor increases.

- Execution Time: As the MSE limit (value) decreases the time taken for developing the desired predictor function increases.

What if this stopping condition was ignored and the GP kept developing a better function?

So, if this stopping condition was ignored and the GP further developed a better function then the accuracy of the predictor function developed would be much better but the time taken for this would be more. Although the GP continued to develop a better function, this algorithm has to stop at some point to produce a forecast of the time series and hence this stopping condition has been introduced. So the best way to obtain a better result is to specify a lowest possible MSE limit while monitoring the time taken by the GP to reach the MSE limit.

3.7 Load Balancing

How are the available resources (processors) divided into the three parts of the parallel implementation?

Let n be the total number of processors available with zero-based indexing. Then the *n-1*th processor acts as the validation part and the *n-2*th processor acts as the Central part. Then the remaining processors between $[0, n-2)$ work for the operational part which divides the total number of tasks as follows.

Suppose there are tasks with indexes 1, 2… m; where *m* is the maximum number of tasks, then each processor with index *p* is allocated tasks with indexes *start_index* and *end_index* inclusive.

$$start\_index = \left\lfloor \frac{m \, * \, p}{(n-2)+1} \right\rfloor ; \quad end\_index = \left\lfloor \frac{m \, * (p+2)}{(n-2)} \right\rfloor ;$$

$$where \text{ p } = \text{ current processor id}$$

The above-mentioned load balancing criteria has been inspired from the example in [23 | page 79].

For example, let n = 9 and the indexes of tasks be [512, 648]. Then index of validation_node = 8 & central_node = 7. The operational nodes have indexes [0, 7).

```
On node: 0 | Start_index: 512 | End_index: 530 | Total tasks on this node: 19
On node: 1 | Start_index: 531 | End_index: 550 | Total tasks on this node: 20
On node: 2 | Start_index: 551 | End_index: 569 | Total tasks on this node: 19
On node: 3 | Start_index: 570 | End_index: 589 | Total tasks on this node: 20
On node: 4 | Start_index: 590 | End_index: 608 | Total tasks on this node: 19
On node: 5 | Start_index: 609 | End_index: 628 | Total tasks on this node: 20
On node: 6 | Start_index: 629 | End_index: 648 | Total tasks on this node: 20
```

CHAPTER IV

EXPERIMENTATION RESULTS & CONCLUSION

4.1 Experimentation Results

In short, the implemented model in this thesis predicts the forecast of a time series by developing an equation that takes the values for a concordant period as input. The output of this equation is the predicted forecast of the time series i.e. the output values are extrapolated from the values in the concordant period. Hence the quality of forecast depends on the concordant period chosen, therefore the best forecast can be obtained when the best concordant period is chosen.

The implemented model in this thesis was evaluated using stock market data for S&P, NASDAQ and NYSE COMPOSITE (DJ). The data set (base period & concordant period) used in this implementation from the above said companies stock market data is of 200 days in length. In the experiments, the prediction was performed for 5 days past the latest day in the historic data. The prediction comparison tables shown below shows the comparison of predicted values with the actual values in the concordant period that was chosen (the prediction has been obtained by extrapolation of these values). Here the direction of actual values is compared to the direction of predicted values. That is if the actual stock value increases or decreases compared to its previous day value and also if the predicted value also increases or decreases accordingly then the direction is referred to be *same* else *different*. If the obtained direction is *same* then the developed equation is valid.

The results shown below are for the parallel model implemented in this thesis. The table shown below compares the 5 day predicted values with the corresponding concordant values of S&P historic stock data from January 3rd 1950 to July 31st 2015.

Date	Actual value	Predicted value	Direction
August 3rd, 2015	103.589996	2115.795445	-
August 4th, 2015	104.110001	2119.107448	*same*
August 5th, 2015	103.540001	2115.476975	*same*
August 6th, 2015	103.830002	2117.324188	*same*
August 7th, 2015	103.980003	2118.279546	*same*

Table-1: 5 days Prediction for S&P data from January 3rd 1950 to July 31st 2015

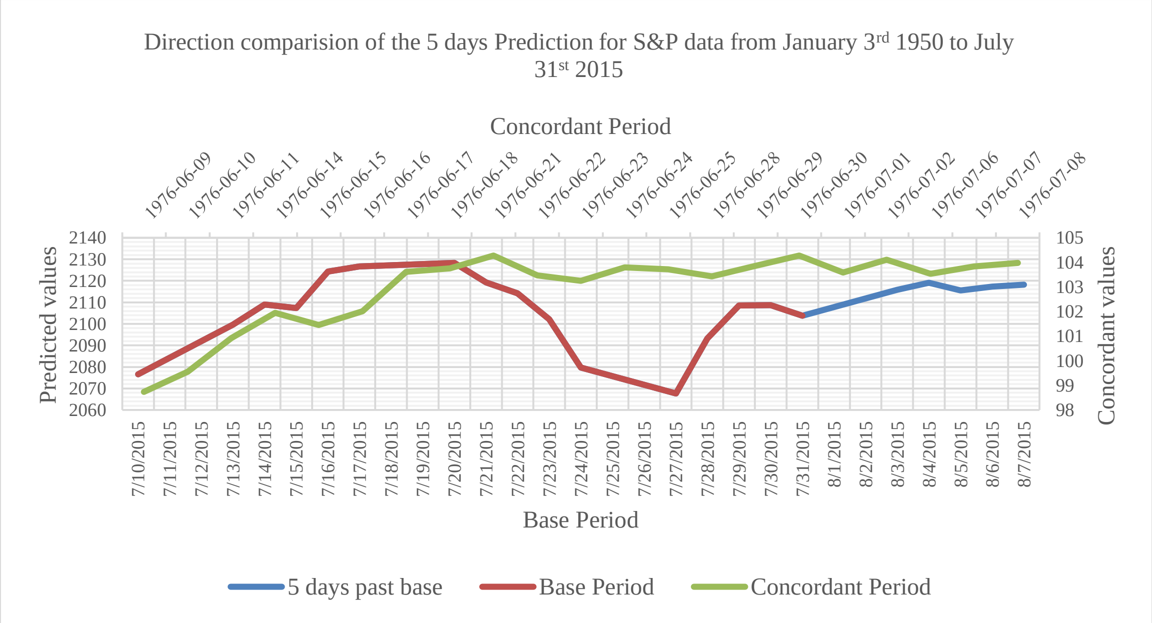


Figure-13: Direction comparison of the 5 days Prediction for S&P data from January 3rd 1950 to July 31st 2015

Figure 13 shows the 5 days prediction for S&P stocks i.e. from 3rd August 2015 to 7th August 2015. The red graph line represents the base period, the green graph line represents the concordant period extended to 5 more days past the concordant period for comparison with the predicted values. The blue graph line represents the forecast extrapolated based on the values from the extended concordant period. Observing the green graph line and the blue graph line, it is understandable that the forecast is in the same direction as in the concordant period. This implies that the desired equation for prediction has been successfully developed.

The table shown below compares the predicted values with the concordant values of S&P historic stock data for next 15 days after base period.

Date	**Actual value**	**Predicted value**	**Direction**
August 3rd, 2015	103.589996	2115.796789	-
August 4th, 2015	104.110001	2119.108265	*Same*
August 5th, 2015	103.540001	2115.478376	*Same*
August 6th, 2015	103.830002	2117.325273	*Same*
August 7th, 2015	103.980003	2118.280482	*Same*
August 10th, 2015	104.980003	2124.647021	*Same*
August 11th, 2015	105.900002	2130.501959	*Same*
August 12th, 2015	105.669998	2129.038401	*Same*
August 13th, 2015	105.949997	2130.820068	*Same*
August 14th, 2015	105.199997	2126.047276	*Same*
August 17th, 2015	104.68	2122.737311	*Same*
August 18th, 2015	104.290001	2120.254372	*Same*
August 19th, 2015	103.720001	2116.624747	*Same*
August 20th, 2015	103.82	2117.261578	*Same*
August 21st, 2015	103.93	2117.962068	*Same*

Table-2: 15 days Prediction for S&P data from January 3rd 1950 to July 31st 2015

By observing the graph shown in figure-14, it is understandable that the direction of prediction and the concordance period is same all over. This implies that the implemented parallel forecasting model works well for prediction more number of days.

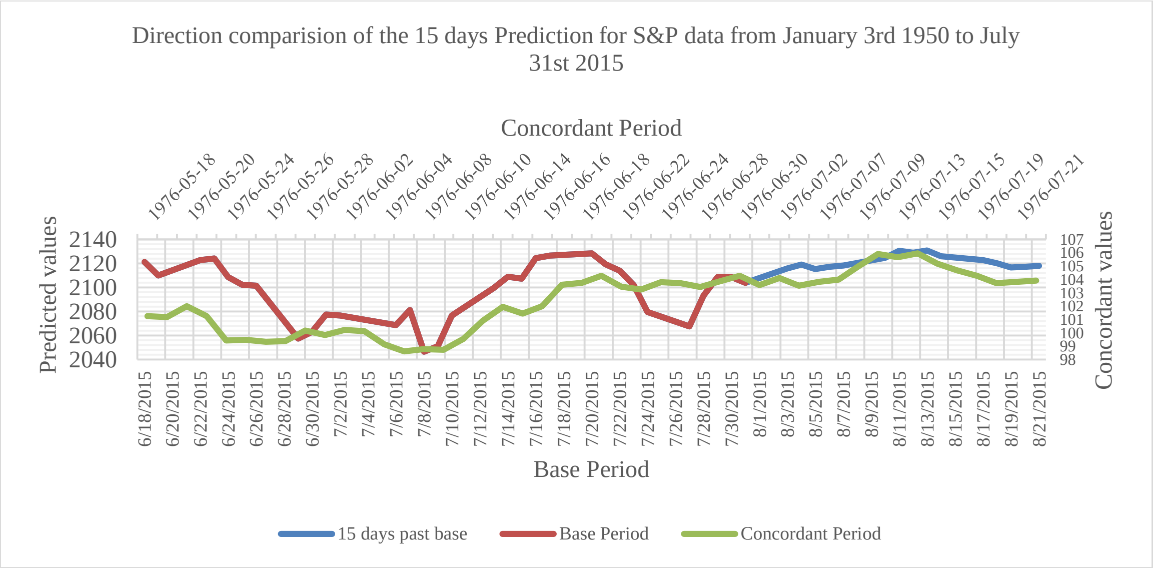


Figure-14: Direction comparison of the 15 days Prediction for S&P data from January 3rd 1950 to July 31st 2015

Another experiment was conducted with different range of S&P stock history i.e. from 3rd January 1950 to 3rd June 1999. The table shown below compares the predicted values with the concordant values of S&P historic stock data for next 10 days after base period. It is again observed from the graph shown in figure-15 that the predicted values are in the same direction as the concordant period.

Date	Actual value	Predicted value	Direction
June 4th, 1999	70.040001	1349.329352	-
June 7th, 1999	69.410004	1334.217167	*Same*
June 8th, 1999	69.07	1326.061249	*Same*
June 9th, 1999	69.370003	1333.257634	*Same*
June 10th, 1999	68.860001	1321.023854	*Same*
June 11th, 1999	69.459999	1335.416432	*Same*
June 14th, 1999	69.940002	1346.930605	*Same*
June 15th, 1999	70.220001	1353.647139	*Same*
June 16th, 1999	69.739998	1342.132967	*Same*
June 17th, 1999	70.040001	1349.329352	*Same*

Table-3: 10 days prediction for S&P data from January 3rd 1950 to June 3rd 1999

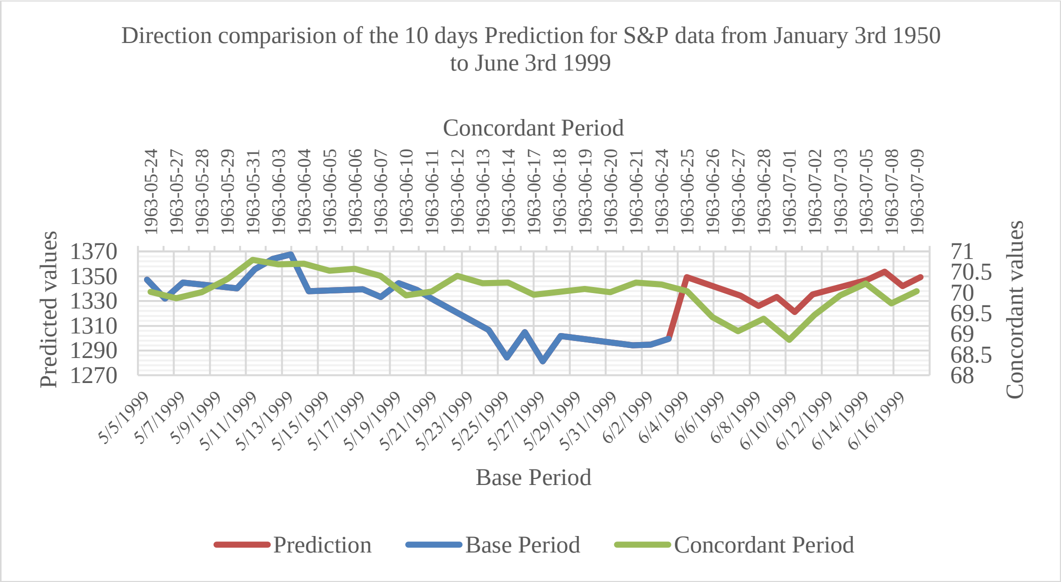


Figure-15: Direction comparison of the 10 days Prediction for S&P data from January 3rd 1950 to June 3rd 1999

A similar experiment was performed on the NASDAQ historic stock data. The table shown below compares the predicted values with the concordant values of NASDAQ historic stock data from February 5th, 1971 to July 31st, 2015.

Date	Actual value	Predicted value	Direction
August 3rd, 2015	4165.609863	5194.377099	-
August 4th, 2015	4156.189941	5189.23252	*Same*
August 5th, 2015	4174.669922	5199.314773	*Same*
August 6th, 2015	4113.299805	5165.668646	*Same*
August 7th, 2015	4183.02002	5203.856546	*Same*

Table-4: 5 days prediction for NASDAQ data from February 5th 1971 to July 31st 2015

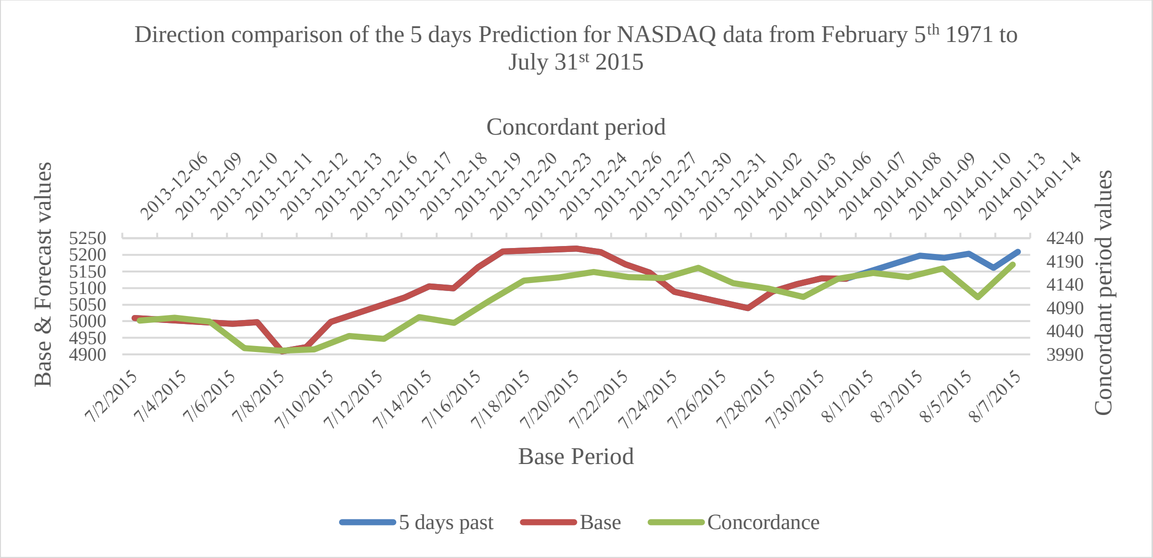


Figure-16: Direction comparison of the 5 days Prediction for NASDAQ data from February 5th 1971 to July 31st 2015

Figure 16 shows the 5 day prediction for NASDAQ stocks i.e. from 3rd August 2015 to 7th August 2015. The red graph line represents the base period, the green graph line represents the concordant period extended to 5 more days past the concordant period. The blue graph line represents the forecast extrapolated based on the values from the extended concordant period. Observing the green graph line and the blue graph line, it is understandable that the forecast is in the same direction as in the concordant period. Again this implies that the desired equation for prediction has been developed.

The table shown below compares the predicted values with the concordant values of NASDAQ historic stock data for next 15 days after base period. It is observed from its corresponding graph in figure-17 that the direction of prediction and the concordance period is same all over.

Date	**Actual value**	**Predicted value**	**Direction**
August 3rd, 2015	4165.609863	5208.259059	-
August 4th, 2015	4156.189941	5202.057049	*Same*
August 5th, 2015	4174.669922	5214.224137	*Same*
August 6th, 2015	4113.299805	5173.818487	*Same*

August 7th, 2015	4183.02002	5219.721783	*same*
August 10th, 2015	4214.879883	5240.698091	*same*
August 11th, 2015	4218.689941	5243.206606	*same*
August 12th, 2015	4197.580078	5229.308022	*same*
August 13th, 2015	4225.759766	5247.861329	*same*
August 14th, 2015	4243	5259.212177	*same*
August 17th, 2015	4218.879883	5243.331663	*same*
August 18th, 2015	4128.169922	5183.608867	*same*
August 19th, 2015	4083.610107	5154.271	*same*
August 20th, 2015	4097.959961	5163.718842	*same*
August 21st, 2015	4051.429932	5133.083801	*same*

Table-5: 15 days prediction for NASDAQ data from February 5th 1971 to July 31st 2015

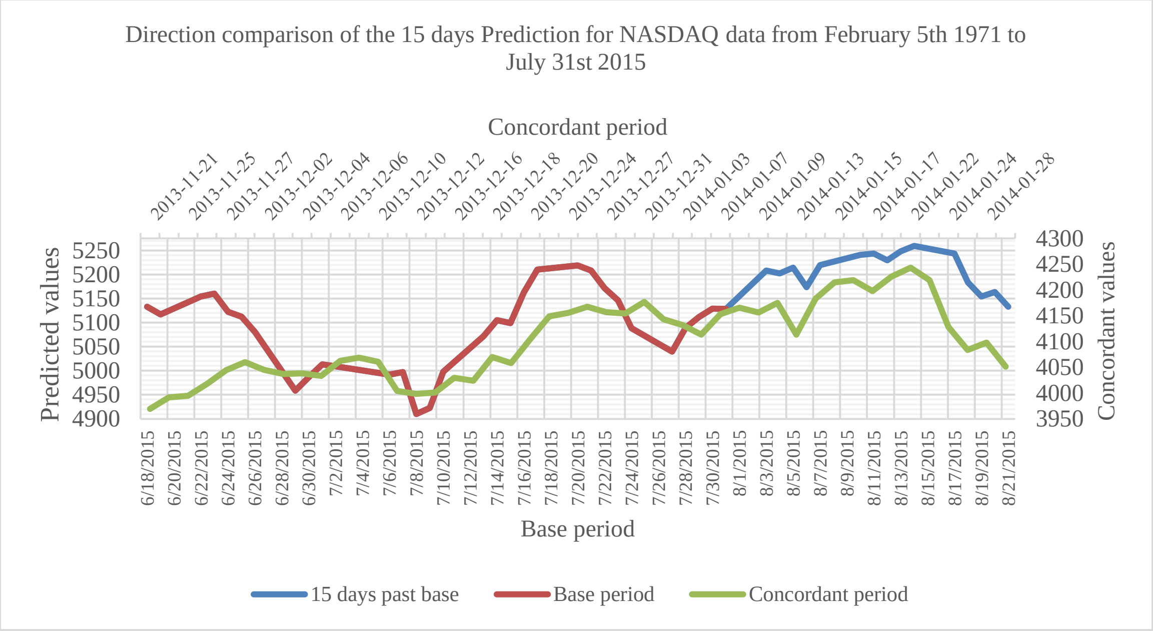


Figure-17: Direction comparison of the 15 days Prediction for NASDAQ data from February 5th 1971 to July 31st 2015

The major purpose of the model implemented in this thesis is to decrease the time taken for this model to forecast the data. This has been done by parallelizing the model proposed in [1]. The parallelization implemented has been tested on a different set of a number of processors to check the behavior of time taken by the algorithm. Since Genetic algorithm which is one of the major component of this forecasting algorithm is based on generating generations randomly, the algorithm was executed seven times and then the average of the total time taken in all executions is considered for comparison. The same experiment has been conducted on different number processors i.e. 12, 18, 24, 30 & 36 processors.

The parallel model implemented in this model has a stopping condition introduced to control the start and stop of the genetic programming which is used to develop the forecasting function. The table shown below shows the time taken for this model to run using S&P historic data on a different number of processors. Observing the values in the table below it is understandable that the time taken for prediction is decreasing as the number of processors used for the execution increases.

# of processors	**Total time is taken in milliseconds**	**Average time in milliseconds**
12	150,997,413	21,571,059.000
18	82,515,770	11,787,967.140
24	71,521,543	10,217,363.290
30	65,231,350	9,318,764.286
36	60,276,523	8,610,931.857

Table-5: Times taken for S&P prediction in parallel implementation

When this experiment was conducted with same data on the serial implementation of this model, the time taken by this implementation is 36,949,380.400 milliseconds which is 15,378,321.400 milliseconds more than the time taken by the parallel implementation using 12 processors.

Similarly, the time taken while performing the same experiment on the NASDAQ historic stock data is shown in the table below. Observing this data again makes it understandable that the time taken for prediction is decreasing as the number of processors used for the execution increases.

# of processors	Total time is taken in milliseconds	Average time in milliseconds
12	12,781,906	1,825,986.571
18	9,775,378	1,396,482.571
24	9,273,270	1,324,752.857
30	6,570,147	938,592.429
36	4,003,944	571,992

Table-6: Times taken for NASDAQ prediction

When this experiment was conducted with same data on the serial implementation of this model, the time taken by this implementation is 3,997,762.800 milliseconds which is 2,171,776.229 milliseconds more than the time taken by the parallel implementation using 12 processors.

4.2 Conclusion

The parallel implementation of forecasting algorithm with added features like allowing the user to choose the parameters like stopping condition value and also making the whole process a batch process makes it easier for the end user to work with this algorithm. Observing the experimentation results we can understand that, parallelization of the forecasting algorithm reduces the amount of time taken for completing the task of this forecasting algorithm.

Since the fusion model proposed earlier in [1] is controlled manually i.e. the user manually started the algorithm and manually stopped the algorithm and hence there was no specific stopping condition for genetic programming. This makes the previous model open bounded and also it is not possible for the user to determine the accuracy of the forecasting function developed.

In the model implemented in this thesis, the user is allowed to specify a mean square error (MSE) limit which is used to check (explained in section 3.6.2) the quality of the forecasting condition developed. This makes the user be more sure about how accurate is the prediction. Based on the MSE limit specified by the user, following two factors are dependent,

- Accuracy: As the MSE (value) limit decreases the accuracy of the predictor (forecasting function) increases.
- Execution Time: As the MSE (value) limit decreases the time taken for developing the desired predictor (forecasting function) increases.

Since the time taken by the model implemented in this thesis cannot be compared with the previous model, the time taken was compared by executing the implemented model on a different number of processors. This comparison shows that the time taken by the implemented model decreases as the number processors used for execution increases. As a future work, this implemented model can be improved by introducing more concordance computation methods to find a much better concordant segment of the time series for prediction.

REFERENCES

[1] Mahesh S. Khadha, Benjamin Popp, K.M. George, N Park: “A New approach for time series forecasting based on genetic algorithm”, CAINE 2010

[2] Mujahid Tabassum and Kuruvilla Mathew, "A Genetic Algorithm Analysis towards Optimization solutions", International Journal of Digital Information and Wireless Communications (IJDIWC) 4(1): 124-142, The Society of Digital Information and Wireless Communications, 2014 (ISSN: 2225-658X)

[3] William B. Langdon, Riccardo Poli, Nicholas F. McPhee, & John R. Koza, “Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications”, Studies in Computational Intelligence (SCI) 115, 927–1028 (2008).

[4] Kendall, M., A New Measure of Rank Correlation,”Biometrika”, Vol. 30, pp. 81-89, 1938

[5] Oklahoma State University High-Performance Computing Center: http://hpc.it.okstate.edu/

[6] William Gropp, Ewing Lusk, Nathan Doss, Anthony Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard", Parallel Computing 22 (1996) 789-828

[7] Mark Baker, Bryan Carpenter, Aamir Shaf, "MPJ Express: Towards Thread Safe Java HPC", Cluster Computing, 2006 IEEE International Conference on 25-28 Sept. 2006

[8] Aamir Shafi, Jawad Manzoor, Kamran Hameed, "Multicore-enabling the MPJ express messaging library", PPPJ '10 Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, Pages 49-58

[9] MPJ Express User Guide, http://mpj-express.org/docs/guides/linuxguide.pdf

[10] M. Oussaidene, B. Chopard, O. V. Pictet, M. Tomassini: “Parallel Genetic Programming: an application to Trading Models Evolution”, “Parallel Computing, 23:1183-1198, 1997”

[11] Fernandez F., Tomassini M., Punch III, W. F., Sanchez J. M., “Experimental Study of multipopulation parallel genetic programming”, “EuroGP 2000, LNCS 1802, pp. 283-293, 2000”

[12] F. Fernandez, M. Tomassini, L. Vanneschi, L. Bucher: “A Distributed Computing Environment for Genetic Programming Using MPI”, “EuroPVM/MPI 2000, LNCS 1908, pp. 322-329, 2000”

[13] S. Baluja, “The evolution of genetic algorithms: Towards massive parallelism”, “in Proceedings of the Tenth International Conference on Machine Learning, San Mateo, CA, 1993, pp. 1-8, Morgan Kaufmann”.

[14] T. Starkweather, D. Whitley, and K. Mathias, “Optimization using distributed genetic algorithms”, “Parallel Problem Solving from Nature, volume 496 of Lecture Notes in Computer Science, pp. 176-185, Heidelberg, 1991”

[15] Jing-Jun Zhang, Wen-Juan Liu, Guang Yuan Liu: “Parallel Genetic Algorithm Based on the MPI Environment”, “TELKOMNIKA, Vol. 10, No. 7, November 2012, pp. 1708-1715 e-ISSN: 2087-278X”

[16] Janko Strabburg, C. Gonzalez-Martel, V. Alexandrov: “Parallel genetic algorithms for stock market trading rules”, “International Conference on Computational Science”, ICCS 2012

[17] E. Cantu-Paz, “A Survey of Parallel Genetic Algorithms”, “Technical Report, IlliGAL 97003, University of Illinois at Urbana-Champaign, 1997”

[18] P. Adamidis, “Review of parallel genetic algorithms bibliography”, “Technical Report version 1, Aristotle University of Thessaloniki, Thessaloniki, Greece, 1994”

[19] M. Nowostawski, R. Poli: “Parallel Genetic Algorithm Taxonomy”, “Knowledge-Based Intelligent Information Engineering Systems, 1999. Third International Conference”

[20] Y. K. Kwon, B. R. Moon, “Daily Stock Prediction Using Neuro-genetic Hybrids”, GECCO 2003, LNCS 2724, pages 2203-2214, 2003

[21] M. Oussaidene, B. Chopard, O. V. Pictet, M. Tomassini: “Parallel Genetic Programming: an application to Trading Models Induction”, “Genetic Programming 1996, pages 357-362. The MIT Press, Cambridge MA, 1996. Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University”

[22] A. Munawar, M. Wasib, M. Munetomo, and K. Akama, “A Survey: Genetic Algorithms and the Fast Evolving World of Parallel Computing”, “in the proceedings of IEEE conference, 2008”

[23] George Em Karniadakis, Robert M. Kirby II, “Parallel Scientific Computing in C++ and MPI”, “published in 2003”

[24] MPJ Express: http://mpj-express.org/

[25] MPJ Express: http://mpjexpress.blogspot.com/

[26] Message Passing Interface Forum: http://www.mpi-forum.org

[27] Genetic Programming Image: http://dave-reed.com/csc550.S03/Presentations/GP.ppt

[28] Computing cluster image: https://en.wikipedia.org/wiki/Computer_cluster; this image was used to recreate and represent the OSU Cowboy High Performance Computing Cluster [5] graphically.

[29] MPJ Express architecture: http://mpj-express.org/software/mpj-design-newest.png

[30] MPI Program structure: https://computing.llnl.gov/tutorials/mpi/

[31] Walter Daelemans, “Machine Learning Approaches”, Syntactic Wordclass Tagging Volume 9 of the series Text, Speech and Language Technology pp 285-304

[32] Christopher Bishop, “Pattern Recognition & Machine Learning”, ISBN: 978-0-387-31073-2

[33] David M. Allen, “Mean Square Error of Prediction as a Criterion for Selecting Variable”, Technometrics, Vol. 13, No. 3 August 1971, pp. 469-475

VITA

SREENIVASA SIVA BHANODHAY NANUGONDA

Candidate for the Degree of

Master of Science

Thesis: PARALLEL IMPLEMENTATION OF A FORECASTING ALGORITHM

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December, 2015.

Completed the requirements for the Bachelor of Technology in Electronics & Communication Engineering at Jawaharlal Nehru Technological University, Hyderabad, India in 2013.

Experience:

Lead developer & Technologist | Legal Market Stack, Cary – NC | May 2014 – Aug 2014

- Worked closely with the CEO. Advisor & decision maker for the technologies and methodologies used in the project.
- Created and managed AWS-EC2 instances to setup development & production servers of the company.
- Worked from scratch in design, development, testing and launching a cloud-based practice management solution for the legal community. This included SCRUM calls and regular strategy management sessions for business development and planning.
- Wrote automation scripts that run on the server side to automatically deploy a completely new, full-featured professional website customized with client's requirements; this automatic deploy process also includes client's Customer Relationship Management tool and email account on the company's CRM-server & email-server respectively. Automatic acknowledgment emails sent.