A Parallel and Concurrent Implementation of Lin-Kernighan Heuristic (LKH-2) for Solving Traveling Salesman Problem for Multi-Core Processors using SPC^3 Programming Model

Muhammad Ali Ismail
Assistant Professor
Dept. of Computer & Info. Sys. Engg.
NED University of Engg. & Tech.
Karachi, Pakistan
maismail@neduet.edu.pk

Dr. Shahid H. Mirza
Professor
Usman Institute of Engg. & Tech.
Karachi, Pakistan
sh_mirza@uit.edu.pk

Dr. Talat Altaf
Professor & Dean (ECE)
Faculty of Elect & Comp Engg.
NED University of Engg. & Tech
Karachi, Pakistan
deanece@neduet.edu.pk

***Abstract*— With the arrival of multi-cores, every processor has now built-in parallel computational power and that can be fully utilized only if the program in execution is written accordingly. This study is a part of an on-going research for designing of a new parallel programming model for multi-core processors. In this paper we have presented a combined parallel and concurrent implementation of Lin-Kernighan Heuristic (LKH-2) for Solving Travelling Salesman Problem (TSP) using a newly developed parallel programming model, SPC^3 PM, for general purpose multi-core processors. This implementation is found to be very simple, highly efficient, scalable and less time consuming in compare to the existing LKH-2 serial implementations in multi-core processing environment. We have tested our parallel implementation of LKH-2 with medium and large size TSP instances of TSBLIB. And for all these tests our proposed approach has shown much improved performance and scalability.**

Keywords- TSP; Parallel Heuristics; Multi-core processors, parallel programming models.

I. INTRODUCTION

Multi-core processors are becoming common and they have built-in parallel computational power and which can be fully utilized only if the program in execution is written accordingly. Most software today is grossly inefficient for multi-core processors, as they are not written for the support of parallelism or concurrency. Writing an efficient and scalable parallel program is now much complex. Scalability embodies the concept that a programmer should be able to get benefits in performance as the number of processor cores increases. Breaking up an application into a few tasks is not a long-term solution. In order to make most of multi-core processors, either, lots and lots of parallelism are actually needed for efficient execution of a program on larger number of cores, or secondly, make a program concurrently executable on multi-cores [1, 2, 3].

The classical Travelling Salesman Problem (TSP) is one of the most representative irregular problems in combinatorial optimization. Despite its simple formulation, TSP is hard to solve. The difficulty becomes apparent when one considers the number of possible tours. For a symmetric problem with 'n' cities there are (n-1)!/2 possible tours. If 'n' is 20, there are more than 10^{18} tours. For 7397-city problem in TSPLIB, there will be more than $10^{25,000}$ possible tours. In comparison it may be noted that the number of elementary particles in the universe has been estimated to be 'only' 10^{87}[5].TSP has diversified application areas because of its generalized nature. TSP is being used to solve many major problems of nearly all engineering disciplines, medicine and computational sciences. [4, 6, 9].

Lin-Kernighan heuristic (LKH) is an implementation of local search optimization meta-heuristic [11, 12] for solving TSP [5, 7, 9, 10]. This heuristic is generally considered to be one of the most effective methods for generating optimal or near-optimal solutions for the symmetric traveling salesman problem. Computational experiments have shown that LKH is highly effective. Even though the algorithm is approximate, optimal solutions are produced with an impressively high frequency. LKH has produced optimal solutions for all solved problems including an 85,900-city instance in TSPLIB. Furthermore, this algorithm has improved the best known solutions for a series of large-scale instances with unknown optima, like 'World TSP' of 1,904,711-city instance. After the original algorithm (LK), its two successive variants LKH-1 and LKH-2 have also been proposed with further improvements in the original algorithm [7, 9, 13].

In this paper we have presented an efficient parallel and concurrent implementation of Lin-Kernighan Heuristic (LKH-2) for Solving Travelling Salesman Problem (TSP) using a newly developed parallel programming model, SPC^3 PM, **S**erial, **P**arallel, and **C**oncurrent **C**ore to **C**ore **P**rogramming **M**odel developed for multi-core processors. It is a serial-like task-oriented multi-threaded parallel programming model for multi-core processors that enables developers to easily write a new parallel code or convert an existing code written for a single processor. The programmer can scale a program for use

with specified number of cores. And ensure efficient task load balancing among the cores [1].

The rest of the paper is organized as follows. In section II, the TSP problem and related solutions are discussed. In subsequent section III, the LKH-2 algorithm and its serial execution are analyzed in order to make it parallel and suitable for multi-core processors using SPC^3 PM. Features and programming with SPC^3 PM are highlighted in section IV. The parallel implementation of LKH-2 based on SPC^{3}PM is presented in section V. In section VI, the experimental setup and results are discussed. Finally, conclusion and future work are given in section VII.

II. TRAVELLING SALESMAN PROBLEM AND RELATED SOLUTION

TSP is one of the most famous, irregular and classical combinatorial optimization problems. It has been proven that TSP is a member of the set of NP-complete problems. In TSP, a salesman is considered who has to visit *n* cities, the TSP asks for the shortest tour through all the cities such that no city is visited twice and the salesman returns at the end of tour back to the staring city.

Mathematically, let $G = (V, E)$ be a graph, where *V* is a set of *n* nodes and *E* is set of arcs. Let $C = [C_{ij}]$ be a cost matrix associated with *E*, where C_{ij} represents the cost of going from city i to city j. The problem is to find a permutation$(i_1, i_2, i_3, \ldots\ldots\ i_n)$ of the integers from 1 through *n* that minimizes the quantity $c_{i_1 i_2} + c_{i_2 i_3} + c_{i_3 i_4} + \ldots\ldots + c_{i_n i_1}$. Using integer programming formulation, the TSP can be defined as

$$c = \min \sum_{i \in V} \sum_{j \in V} c_{i\,j} x_{i\,j}$$

Such that $\sum_{j \in V} x_{i\,j} = 1, i \in V$ And $\sum_{i \in V} x_{i\,j} = 1, j \in V$

$$\sum_{i \in S} \sum_{j \in S} x_{i\,j} \leq |S|-1, \ \forall S \subset V, S \neq \emptyset$$

And $x_{i\,j} \in \{0,1\}, for\ all\ i, j \in V.$

Where $x_{i\,j} = 1$ if arc (*i*,*j*) is in the solution and 0 otherwise.

Properties of the cost matrix C are used to classify problems.

- If $c_{ij} = c_{ji}$ for all i and j, the problem is said to be symmetric; otherwise, it is asymmetric.
- If the triangle inequality holds ($c_{ik} \leq c_{ij} + c_{jk}$ for all i, j and k), the problem is said to be metric.
- If c_{ij} are Euclidean distances between points in the plane, the problem is said to be Euclidean. A Euclidean problem can be both symmetric and metric.

In order to find the optimal solution for any TSP based problem a number of solutions have been proposed, which can be classified into four classes as Exact, Heuristic, Meta-heuristic and hyper heuristics Algorithms.

A. Exact Algorithms

These algorithms are used when we want to obtain an exact optimal solution. In this, every possible solution is identified and compared for optimal solution. These algorithms are suitable for a smaller number of inputs. Brute-force method, Dynamic programming algorithm of Hell and Karp, Branch-and-Bound and Branch-and-Cut algorithm are some of the famous algorithms of this class [4, 5].

B. TSP Heuristics:

These heuristics are used when the problem size is large enough, time is limited or the data of the instance is not exact. In this class, instead of finding all possible solutions of a given problem, a sub optimal solution is identified. TSP heuristic can be roughly partitioned into two classes: 'Constructive heuristic' and 'Improvement heuristic'. Constructive heuristics build a tour from scratch and stop when one solution is produced. Improvement heuristics start from a tour normally obtained using a construction heuristic and iteratively improve it by changing some parts of it at each iteration. Improvement heuristics are typically much faster than the exact algorithm and often produce solutions very close to the optimal one. Greedy Algorithms, Nearest Neighbor, Vertex Insertion, Random Insertion, Cheapest Insertion, Saving Heuristics, Christofides Heuristics, Krap-Steele Heuristics, and ejection-chain method are the well known proposed heuristics algorithm of this class [4, 5, 6].

C. Meta-Heuristics

These are intelligent heuristics algorithms having the ability to find their way out of local optima. The Meta-heuristic approaches are the combination of first two classes. These Meta-heuristics contain implicit intelligent algorithms, ability to find their way out of local optima and possibility of numerous variants and hybrids. These heuristics are relatively more challenging to parallelize. Due to these reasons meta-heuristic approaches have drawn attention of many researchers.

Many of the well-known meta-heuristics have been proposed like Random optimization, Local search optimization, Greedy algorithm and hill-climbing, Best-first search, Genetic algorithms, Simulated annealing, Tabu search, Ant colony optimization, Particle swarm optimization, Gravitational search algorithm, Stochastic diffusion search, Harmony search, Variable neighborhood search, Glowworm swarm optimization (GSO) and Artificial Bee colony algorithm. However because of TSP nature, all these meta-heuristics cannot be used for solving TSP. Specific meta-heuristics used for solving TSP include Simulated Annealing, Genetic Algorithms, Neural Networks, Tabu Search, Ant colony optimization, and Local search optimization [4, 5, 6].

D. Hyper-Heuristics

This is an emerging direction in modern search technology. It is termed as Hyper-heuristic as it aims to raise the level of granularity at which optimization system can operate. They are broadly concerned with intelligently choosing the right heuristic or algorithm in given situation. A hyper-heuristic works at a higher level when compared with the typical application of meta-heuristics to optimize problems, i-e; a

hyper-heuristics could be taken as a heuristic or meta-heuristic which operates on other low level heuristics or meta-heuristics [4].

III. LIN-KERNIGHAN HEURISTIC AND SERIAL EXECUTION OF LKH-2 SOFTWARE

The Lin–Kernighan algorithm belongs to the class of so-called local search algorithms [5, 7, 9, 10]. A local search algorithm starts at some location in the search space and subsequently moves from the present location to a neighboring location. LKH has produced optimal solutions for all solved problems including an 85,900-city instance in TSPLIB. Furthermore, this algorithm has improved the best known solutions for a series of large-scale instances with unknown optima, like 'World TSP' of 1,904,711-city instance [5, 13].

The algorithm is specified in exchanges (or moves) that can convert one candidate solution into another. Given a feasible TSP tour, the algorithm repeatedly performs exchanges that reduce the length of the current tour, until a tour is reached for which no exchange yields an improvement. This process may be repeated many times from initial tours generated in some randomized way.

The Lin–Kernighan algorithm (LK) performs so-called k-opt moves on tours. A k-opt move changes a tour by replacing k edges from the tour by k edges in such a way that a shorter tour is achieved. Let T be the current tour. At each iteration step the algorithm attempts to find two sets of edges, $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_k\}$, such that, if the edges of X are deleted from T and replaced by the edges of Y , the result is a better tour. The edges of X are called out-edges. The edges of Y are called in-edges. The detail of LKH-2 software can be found in [7].

A. LKH-2 Software

LKH-2 software provides an effective serial implementation of the Lin-Kernighan heuristic Algorithm with General k-opt Sub-moves for solving the traveling salesman problem. It is written in visual C++. Computational experiments have shown that LKH-2 software is highly effective for solving TSP. This software has produced optimal solutions for all solved problems we have been able to obtain including a 85,900-city instance available in the TSPLIB. Furthermore, it has improved the best known solutions for a series of large-scale instances with unknown optima, among these a 1,904,711-city instance commonly known as World TSP. Similarly LKH-2- software also currently holds the record for all instances with unknown optima provided in the DIMACS TSP Challenge which provides many benchmark instances range from 1,000 to 10,000,000 cities. Its six versions 2.0.0, 2.0.1, 2.0.2, 2.0.2, 2.0.3, 2.0.4 and 2.0.5 have been released. For our study we have used its latest 2.0.5 version released in November 2010. This software can be downloaded free from [13].

B. Execution of LKH-2 software

For converting the serial LKH-2 software into parallel and make it suitable for multi-core processors the LKH heuristics and its LKH-2 software execution were analyzed in detail. On analyzing it is found that LKH-2 Software is written using functional programming. Its complex computation is divided into ninety eight functions which can be called from the main program accordingly. This software also takes help of thirteen header files. On further exploration it was found that working of LKH-2 software can be broken into seven basic stages which may help in its parallelization. All the seven stages are discussed below. A flow chart representing the stages of LKH-2 software on the basis of the stages is shown in Fig. 1.

1) Stage 1: Read parameter file: This is the first step in LKH-2 software. A function is calleld to open the parameter file and to read the specified problem parameters in the file.

2) Stage 2: Read Problem file: In the next step, the specifed problem file is read. In the TSP library all the instances and their releated infromation is placed in an indivuall files using a standard format. This file is known as the problem file. The "ReadProblem function" in LKH-2 software reads the problem data in TSPLIB format for further processing.

3) Stage 3: Partitioning of the problem: After reading the problem, the large problem may be divided into number of sub-problems as defined in the parameter file using the parameter 'sub-problem size'. If sub-problem size is zero than no partitioning of the problem is done. Else by default the sub-problems are determined by sub-dividing the tour into segments of equal size. However LKH-2 software also provides five other different techniques to partition the problem. These include Delaunay Partitioning, Karp Partitioning, K-Means Partitioning, Rohe Partitioning and MOORE or SIERPINSKI Partitioning.

4) Stage 4: Initialization of data structures and statistics variable: After reading the probelm and its partitioning, if done, the releated data structures and statistics variables are initialized. The major statistical variables include minimum and maximum trials, total number and number of success trails, minimum and maximum cost, total Cost, minimum and maximum Time, total Time.

5) Stage 5: Generation of Initial Candidate Set: The "CreateCandidateSet" function and its sub-functions determines a set of incident candidate edges for each node. If the penalties (the Pi-values in the paramenter file) is not defined, the "Ascent function" is called to determine a lower bound on the optimal tour using sub-gradient optimization. Else the penalties are read from the file, and the lower bound is computed from a minimum 1-tree. The function "GenerateCandidates" is called to compute the Alpha-values and a set of incident candidate edges is associated to each node.

6) Stage 6: Find Optimal Tour : This is main processing step where the optimal tour is found. After the creation of candidate set, the "FindTour" function' is called 'for predetermined number of times (Runs). FindTour performs a number of trials, where in each trial it attempts to improve a chosen initial tour using the modified Lin-Kernighan edge exchange heuristics. If tour found is better than the existing tour, the tour and time are recorded.

7) Stage 7: Update Statistics: This step is called in two levels. Firstly this step is processed after every individual call for "FindTour" function to update the respective statistical variables. Finally it is processed at the end of total runs of "FindTour" function' to calculate and report the average statistics.

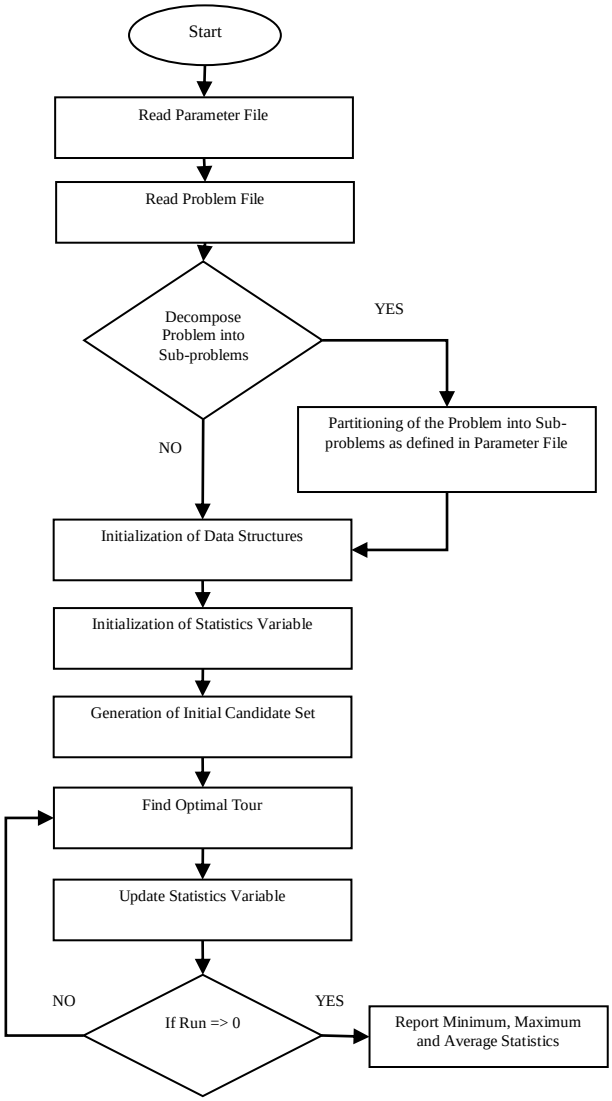

Figure1. Stages in Original serial LKH-2 software

IV. SPC^3 PM (**S**ERIAL, **P**ARALLEL, AND **C**ONCURRENT **C**ORE TO **C**ORE **P**ROGRAMMING **M**ODEL)

SPC^3 PM, (**S**erial, **P**arallel, **C**oncurrent **C**ore to **C**ore **P**rogramming **M**odel), is a serial-like task-oriented multi-threaded parallel programming model for multi-core processors, that enables developers to easily write a new parallel code or convert an existing code written for a single processor. The programmer can scale it for use with specified number of cores. And ensure efficient task load balancing among the cores [1].

SPC^3 PM is motivated with an understanding that existing general-purpose languages do not provide adequate support for parallel programming. Existing parallel languages are largely targeted to scientific applications. They do not provide adequate support for general purpose multi-core programming whereas SPC^3 PM is developed to equip a common programmer with multi-core programming tool for scientific and general purpose computing. It provides a set of rules for algorithm decomposition and a library of primitives that exploit parallelism and concurrency on multi-core processors. SPC^3 PM helps to create applications that reap the benefits of processors having multiple cores as they become available.

SPC^3 PM provides thread parallelism without the programmers requiring having a detailed knowledge of platform details and threading mechanisms for performance and scalability. It helps programmer to control multi-core processor performance without being a threading expert. To use the library a programmer specifies tasks instead of threads and lets the library map those tasks onto threads and threads onto cores in an efficient manner. As a result, the programmer is able to specify parallelism and concurrency far more conveniently and with better results than using raw threads.. The ability to use SPC^3 PM on virtually any processor or any operating system with any C++ compiler also makes it very flexible.

SPC^3 PM has many unique features that distinguish it with all other existing parallel programming models. It supports both data and functional parallel programming. Additionally, it supports nested parallelism, so one can easily build larger parallel components from smaller parallel components. A program written with SPC^3 PM may be executed in serial, parallel and concurrent fashion. Besides, it also provides processor core interaction to the programmer. Using this feature a programmer may assign any task or a number of tasks to any of the cores or set of cores.

A. *Key Features*

The key features of SPC^3 are summarized below.

- SPC^3 is a new shared programming model developed for multi-core processors.
- SPC^3 PM works in two steps: defines the tasks in an application algorithm and then arranges these tasks on cores for execution in a specified fashion.
- It provides Task based Thread-level parallel processing.
- It helps to exploit all the three programming execution approaches, namely, Serial, Parallel and Concurrent.
- It provides a direct access to a core or cores for maximum utilization of processor.
- It supports major decomposition techniques like Data, Functional and Recursive.
- It is easy to program as it follows C/C++ structure.
- It can be used with other shared memory programming model like OpenMP, TBB etc.
- It is scalable and portable.
- Object oriented approach

B. *Programming with SPC^3 PM*

SPC^3 PM provides a higher-level, shared memory, task-based thread parallelism without knowing the platform details

and threading mechanisms. This library can be used in simple C / C++ program having tasks defined as per SPC[3] PM Task Decomposition rules. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner. The result is that SPC[3] PM enables you to specify parallelism and concurrency far more conveniently, and with better results, than using raw threads.

Programming with SPC[3] is based on two steps. First describing the tasks as it specified rules and then programming it using SPC[3] PM Library.

1) Steps involved in the development of an application using SPC[3]PM

- The user determines that his application can be programmed to take advantage of multi-core processors.
- The problem is decomposed by the user following the SPC3 PM 'Task Decomposition Rules'.
- Each Task is coded in C /C++ as an independent unit to be executed independently and simultaneously by each core.
- Coding of Main Program using SPC[3] PM Library to allow the user to run the program in serial, parallel or concurrent mode.
- Compilation of code using any standard C/C++ compiler.
- Execution of Program on a multi-core processor

2) Rules for Task Decomposition

The user can decompose the application / problem on the basis of following rules.

- The user should be able to breakdown the problem in various parts to determine if they can exploit Functional, Data or Recursive decomposition.
- Identify the loops for the loop parallelism and may be defined as Tasks.
- Identify independent operations that can be executed in parallel and may be coded as independent Tasks.
- Identify the large data sets on which single set of computations have to be performed. Target these large data sets as Tasks.
- Tasks should be named as Task1, Task2,….. TaskN. If a Task returns a value it should be named with suffix 'R' like TaskR1, TaskR2…. TaskRN.
- There is no limit on the number of Tasks.
- Each Task should be coded using either C/C++/VC++/C# as an independent function.
- A Task may or may not return the value. A Task should only intake and return structure pointer as a parameter. Initialize all the shared or private parameters in the structure specific to a Task. This structure may be shared or private.
- Arrange the tasks using SPC[3] PM Library in the main program according to the program flow.

3) Program Structure

```
Define Task1
Define Task2
Define Task3
Define Task4
..
Define Task.N

Structure                Structure                Structure                Structure
STRUCT_NAME              STRUCT_NAME              STRUCT_NAME              STRUCT_NAME
{                        {                        {                        {
//The structure          //The structure          //The structure          //The structure
//having private         //having private         //having private         //having private
//or global              //or global              //or global              //or global
//parameters             //parameters             //parameters             //parameters
//associated with a      //associated with a      //associated with a  ..... //associated with a
//specified task         //specified task         //specified task         //specified task
}                        }                        }                        }
STRUCT_NAME              STRUCT_NAME              STRUCT_NAME              STRUCT_NAME
*P_TASK1                 *P_TASK2                 *P_TASK3                 *P_TASK.N

Task1(LPVOID)            Task2(LPVOID)            Task3(LPVOID)            Task.N(LPVOID)
{                        {                        {                  ..... {
//performing             //performing             //performing             //performing
//some                   //some                   //some                   //some
//computation            //computation            //computation            //computation
}                        }                        }                        }

void main( void)
{
    // any declaration;
    // any piece of code :
    Serial (Task1, P_TASK1 );          //execution of task 1 in serial
    // Any other code ;
    Parallel (Task2, P_TASK2);         //execution of task 2 in parallel
    // any other code ;
    Concurrent (Task3 , P_TASK3, Task4, P_TA SK4); //execution of task 3 and 4 concurrently
}
```

C. SPC[3] PM Library

SPC[3] PM provides a set of specified rules to decompose the program into tasks and a library to introduce parallelism in the program written using c/ c++. The library provides three basic functions.

- Serial
- Parallel
- Concurrent

1) Serial: This function is used to specify a Task that should be executed serially. When a Task is executed with in this function, a thread is created to execute the associated task in sequence. The thread is scheduled on the available cores either by operating system or as specified by the programmer. This function has three variants. Serial (Task i) {Basic}, Serial (Task i, core) {for core specification} and *p Serial (Task i, core, *p) {for managing the arguments with core specification}

2) Parallel: This function is used to specify a Task that should be executed in parallel. When a Task is executed with in this function, a team of threads is created to execute the associated task in parallel and has an option to distribute the work of the Task among the threads in a team. These threads are scheduled on the available cores either by operating system

or as specified by the programmer. At the end of a parallel function, there is an implied barrier that forces all threads to wait until the work inside the region has been completed. Only the initial thread continues execution after the end of the parallel function. The thread that starts the parallel construct becomes the master of the new team. Each thread in the team is assigned a unique thread id to identify it. They range from zero (for the master thread) up to one less than the number of threads within the team. This function has also four variants. Parallel (Task i) {Basic}, Parallel (Taski ,num-threads) {for defining max parallel threads}, Parallel (Task i, core list) {for core specification} and *p parallel (Task i, core, *p) {for managing the arguments with core specification}

3) Concurrent: This function is used to specify the number of independent tasks that should be executed in concurrent fashion on available cores. These may be same tasks with different data set or different tasks. When the Tasks are executed defined in this function, a set of threads equal or greater to the number of tasks defined in concurrent function is created such that each task is associated with a thread or threads. These threads are scheduled on the available cores either by operating system or specified by the programmer. in other words , this function is an extension and fusion of serial and parallel functions. All the independent tasks defined in concurrent functions are executed in parallel where as each thread is being executed either serially or in parallel. This function has also three variants. Concurrent (Task i, Taskj, ...Task N) {Basic}, Concurrent (Task i, core , Task j , core, ……) {for core specification} and Concurrent (Task i, core , *p, Task j , core, *p ……) {for managing the arguments with core specification}.

V. PARALLELIZATION OF LKH-2 SOFTWARE USING SPC^3 PM

LKH-2 software is a serial code and cannot make most of multi-cores unless modified accordingly. This LKH-2 software code can be made suitable for multi-core processors by introducing parallelism and concurrency in it. Here it is done using SPC^3 PM.

SPC^3 PM, Serial Parallel and Concurrent Core to Core programming Model provides an environment to decompose the application into tasks using its task decomposition rules and then execute these tasks in serial, parallel and concurrent fashion. As LKH-2 software is written in function style so we have to only restructure the some part of the code to make it suitable for SPC^3 PM.

Working of LKH-2 software can be decomposed into seven stages as discussed in section III. Out of seven, the most important and computational intensive stages are its sixth stage that is finding of the optimal tour using LKH-2 algorithm and seventh stage, that is updating of the statistics accordingly. The other related time consuming step is to execute this stage multiple times as defined in the parameter file (runs). The rest of the stages do not demand much of time and computations and can be executed in serially.

LKH-2 software is parallelized by converting its tour finding and related routines (sixth stage) into tasks according to the SPC^3 PM Task decomposition rules and executing them in parallel using parallel function of SPC^3 PM Library. To execute this stage multiple times as defined in the parameter file (runs), concurrent function of SPC^3 PM is used. This concurrent execution enables to execute this stage in parallel on the available cores. This approach of decomposition and execution of LKH-2 software makes it suitable for parallel execution on multi-core processors.

This two level parallel and concurrent execution of stages also makes this LKH-2 software scalable with respect to multiple-cores processors. The available cores are divided into sets equal to number of runs of stage six. Each set execute the stage concurrently and cores in each set execute the single task of finding the optimal tour in parallel. Number of sets and number of cores in each set is calculated using the following equations respectively.

$$\text{Number of Sets (S)} = \text{(Number of Available Cores (N))/(Number of Runs)} \quad (1)$$

$$\text{Number of Cores in Each Set (C)} = \text{(Number of Available Cores (N))/(Number of Sets (S))} \quad (2)$$

For example, on a 24 cores processor with 8 runs of finding the tour task, the total 8 sets with 3 cores each are created. Each individual execution of the task is performed on each set concurrently. Whereas three cores in each set is responsible to execute the task with in a set in parallel.

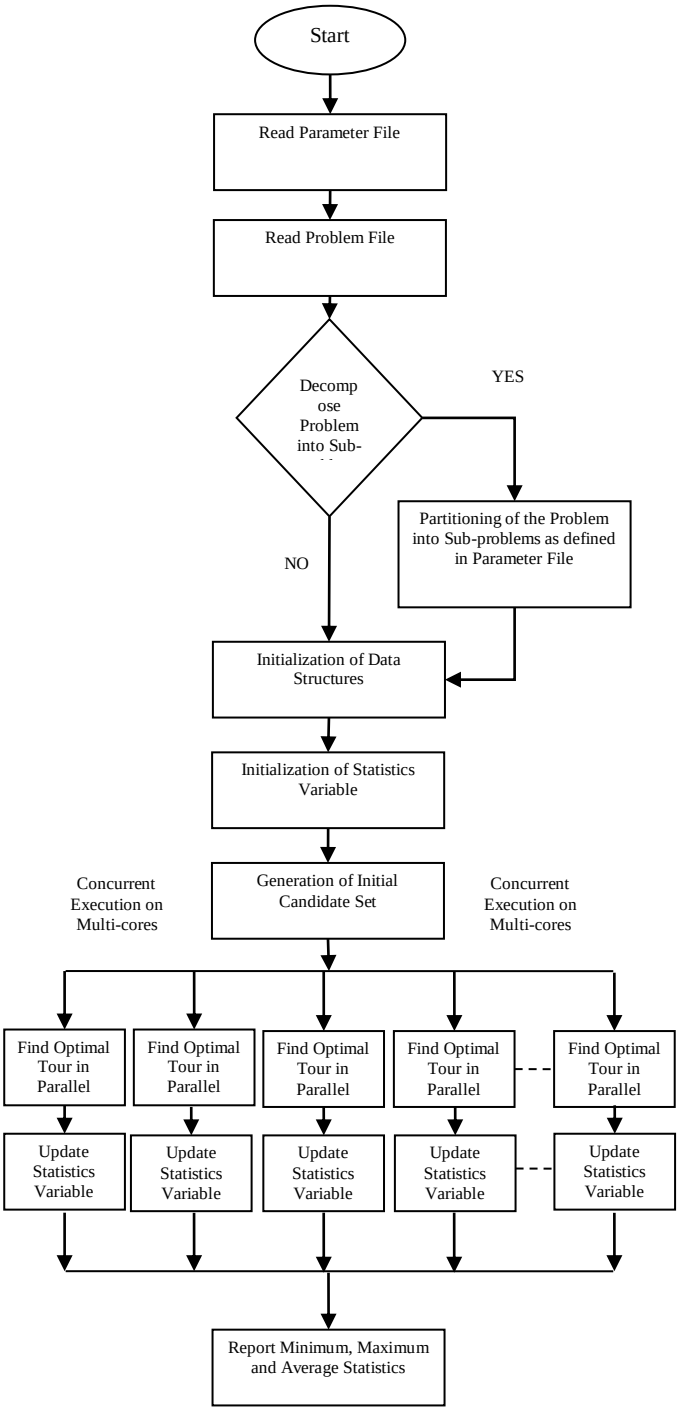

FIGURE2. Stages in parallelized LKH-2 software using SPC^3 PM

VI. PERFORMANCE EVOLUTION

This section discusses the performance comparison of the parallelized LKH-2 software using SPC^3 PM and the original LKH-2 software version 2.0.5. on various instances of TSP library. The LKH-2 is parallelized using SPC^3 PM Task decomposition rules and SPC^3 PM Library to make this serial code suitable for multi-core processors.

A. Experimental Setup

For our study we have selected standard medium size and large size TSP instances of TSBLIB [5, 14, 15]. All the computational tests reported in this section, for both original and parallelized LKH-2 software code with SPC^3 PM, run on the TSPLIB instances, have been made using the default values of parameters defined in LKH-2 software parameter file. These default values have proven to be adequate in many applications [5]. Each TSP instances for both original and parallelized LKH-2 software code with SPC^3 PM, is given ten runs for calculating the optimal tour and for each respective TSP instance, three different execution times, i-e, the minimum execution time out of ten runs, average and total execution time required for all ten runs are then recoded.

For the execution of the algorithms, the latest Intel server 1500ALU with dual six core hyper threaded Intel Xeon 5670 processor is used. Thus total number of parallel threads that can be executed is 2*2*6=24. Operating systems used is 64 bit windows 2008 server.

B. Result Analysis and Observations

Table1 shows the minimum, average and total execution time for original serial LKH-2 software for 10 runs of each medium size TSP instances.

TABLE1. Minimum, Average and Total execution time for original serial LKH-2 software for medium size TSP instances (10 runs each)

TSP Instance	Optimal Value	Average Root Gap	Min. Time (Sec)	Average Time (Sec)	Total Time (Sec)
pr1002	259045	0.00%	1	1	12
si1032	92650	0.00%	5	7	74
u1060	224094	0.01%	54	103	1026
vm1084	239297	0.02%	30	42	420
pcb1173	56892	0.00%	0	3	30
d1291	50801	0.00%	3	4	43
rl1304	252948	0.16%	14	14	140
rl1323	270199	0.02%	2	12	117
nrw1379	56638	0.01%	14	16	158
fl1400	20127	0.18%	3663	3906	39061
u1432	152970	0.00%	3	3	33
fl1577	22204	0.24%	1218	2189	21888
d1655	62128	0.00%	2	4	39
vm1748	336556	0.00%	20	22	220
u1817	57201	0.09%	68	119	1188
rl1889	316536	0.00%	65	135	1348
d2103	79952	0.63%	146	162	1624
gr2121	2707	0.00%	25	30	303
u2319	234256	0.00%	1	1	10
pr2392	378032	0.00%	1	1	10

Table 2 shows the minimum, average and total time for the parallelized LKH-2 software using SPC^3 PM for 10 runs of each medium size TSP instances.

TABLE2: Minimum, Average and Total time for Parallelized LKH-2 software using SPC3 PM for each medium size TSP instances (10 runs each)

TSP Instance	Optimal Value	Average Root Gap	Min. Time (Sec)	Average Time (Sec)	Total Time (Sec)
pr1002	259045	0.00%	0	1	9
si1032	92650	0.00%	2	5	51
u1060	224094	0.01%	34	67	673
vm1084	239297	0.02%	15	27	271
pcb1173	56892	0.00%	0	2	20
d1291	50801	0.00%	2	3	31
rl1304	252948	0.16%	8	10	98
rl1323	270199	0.02%	2	8	77
nrw1379	56638	0.01%	8	11	112
fl1400	20127	0.18%	1883	2370	23695
u1432	152970	0.00%	2	2	22
fl1577	22204	0.24%	809	1422	14222
d1655	62128	0.00%	2	3	28
vm1748	336556	0.00%	11	16	159
u1817	57201	0.09%	44	81	811
rl1889	316536	0.00%	43	100	1001
d2103	79952	0.63%	106	137	1368
gr2121	2707	0.00%	15	22	219
u2319	234256	0.00%	1	1	7
pr2392	378032	0.00%	1	1	7

Following Fig. 3 based on tables 1 and 2, shows the comparison of minimum time between original serial LKH-2 software and parallelized LKH-2 software using SPC^3 PM for the medium size TSP instances. Similarly, Fig. 4 and Fig. 5 show the comparison of average and total time between original serial LKH-2 software and parallelized LKH-2 software using SPC^3 PM for 10 runs of each medium size TSP instances

From Fig. 3, for minimum execution time, it may clearly be observed that our parallelized LKH-2 software using SPC^3 PM requires much lesser time that of the original LKH-2 software requires. It is so because the main function of finding the optimal tour using LKH algorithm is being executed in parallel on the available cores as defined in (2). In this case, 10 runs of each instance are executed concurrently on 20 cores. That is each run has a set of nearly 2 cores for its execution in parallel. Speedup obtained in our case ranges from 1.5 to 1.7, which is where much near to the ideal speedup which should be 2 in this case.

Similarly, from Fig 4, the same observation can be made that the average execution time for parallelized LKH-2 software using SPC^3 PM requires much lesser time that of the original LKH-2 software requires. It is so, because all the required runs of an instance are running in parallel on their respective allocated set of 2 cores.

For the total execution time required for 10 runs of each instances, the parallelized LKH-2 software code shows much greater performance gain in comparison to original LKH-2 code. This is because of the concurrent execution of all required runs on the available cores. In this case as defined by (1), total 10 sets are created. Each set is responsible to execute a run of a given instance. Thus all the runs are executed concurrently on 24 core machine making most of the multi-core processor and reducing the total execution time remarkably. Whereas in serial execution of original LKH-2 software, next run of a TSP instance is executed only after the completion of the first run.

Table 3. shows the minimum, average and total time for original serial LKH-2 software for each large size TSP instances. Similarly, table 4 shows the minimum, average and total time for the parallelized LKH-2 software using SPC^3 PM for each large size TSP instances. All the computational tests reported here are taken with default parameter file and having ten runs for each TSP instance.

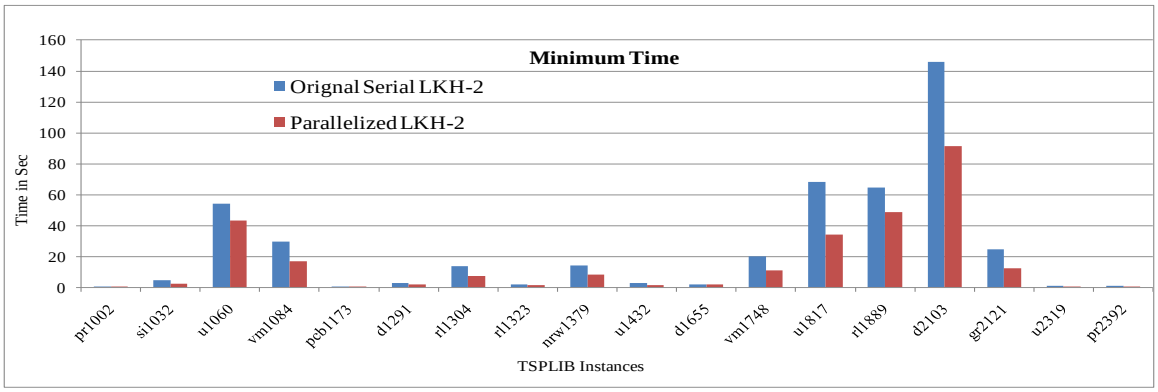

FIGURE 3. Comparison of minimum time between original serial LKH-2 software and parallelized LKH-2 software using SPC^3 PM for the medium size TSP instances calculated for 10 runs

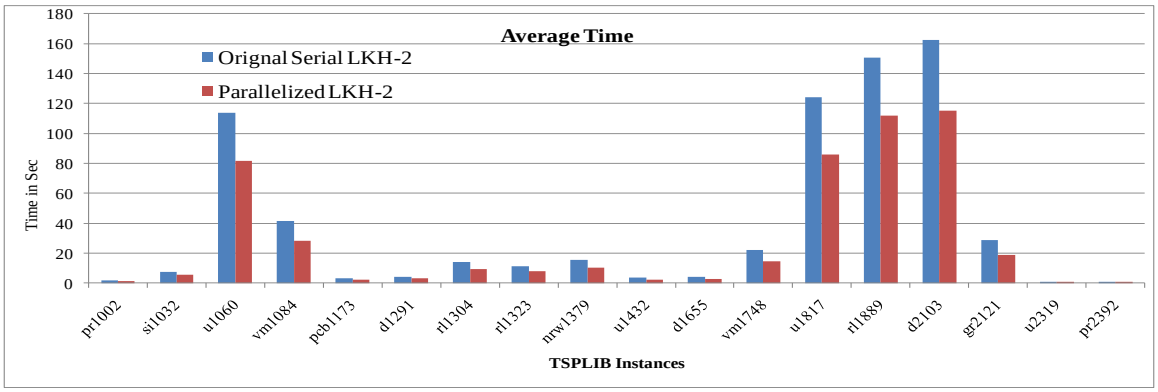

FIGURE 4. Comparison of average time between original serial LKH-2 software and parallelized LKH-2 software using SPC[3] PM for the medium size tsp instances calculated for 10 runs

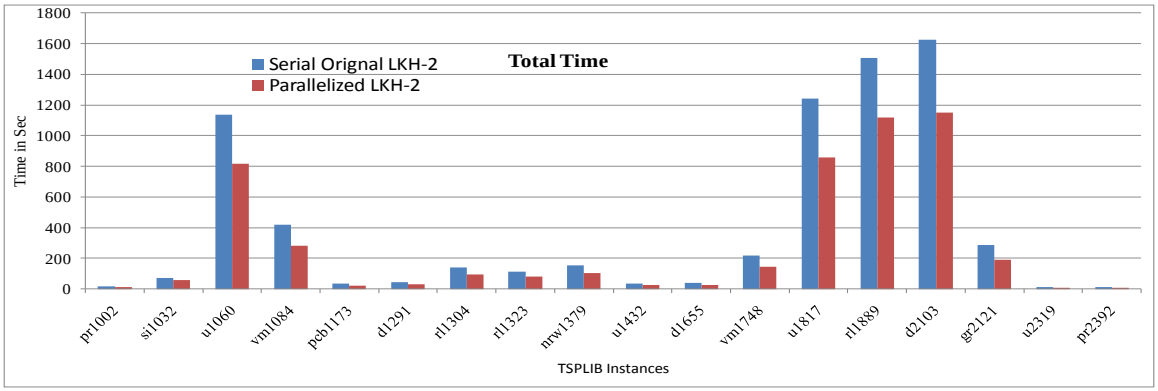

FIGURE 5. Comparison of total time between original serial LKH-2 software and parallelized LKH-2 software using SPC[3] PM for the medium size TSP instances calculated for 10 runs

TABLE 3. Minimum, Average and Total execution time for original serial LKH-2 software for each large size TSP instances (10 runs each)

TSP Instance	Optimal Value	Avg. Root Gap	Min. Time (Sec)	Avg. Time (Sec)	Total Time (Sec)
pcb3038	137694	0.00%	430	499	4993
fl3795	[28723,28772]	0.31%	5114	6473	64725
fnl4461	182566	0.09%	2460	2759	27594
rl5915	[565040,565530]	0.37%	3220	3329	33286
pla7397	23260728	0.00%	1280	1544	15440

TABLE 4. Minimum, Average and Total time for Parallelized LKH-2 software using SPC[3] PM for each large size TSP instances (10 runs each)

TSP Instance	Optimal Value	Avg. Root Gap	Min. Time (Sec)	Avg. Time (Sec)	Total Time (Sec)
pcb3038	137694	0.00%	279	365	540
fl3795	[28723,28772]	0.31%	3299	4474	7109
fnl4461	182566	0.09%	1507	1873	2675
rl5915	[565040,565530]	0.37%	1849	2420	3201
pla7397	23260728	0.00%	762	1107	1525

Following Fig. 6 based on tables 3 and 4 shows the comparison of minimum time between original serial LKH-2 software and parallelized LKH-2 software using SPC[3] PM for the large size TSP instances. Similarly, Fig. 7 and Fig. 8 show the comparison of average and total time between original serial LKH-2 software and parallelized LKH-2 software using SPC[3] PM for the large size TSP instances.

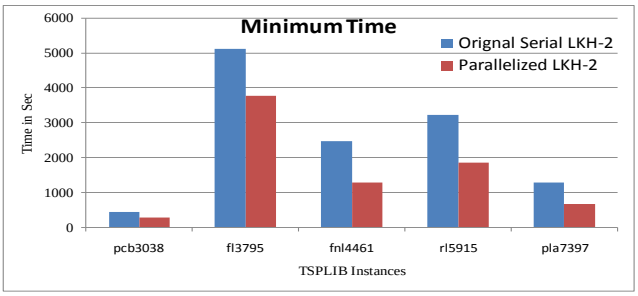

FIGURE 6. Comparison of minimum execution time between original serial lkh-2 software and parallelized LKH-2 software using SPC[3] PM for the large size TSP instance calculated for 10 runs

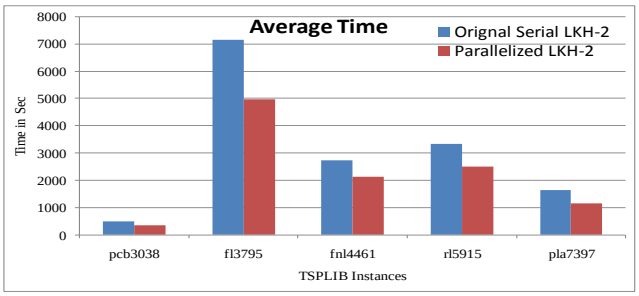

FIGURE 7. Comparison of average execution time between original serial lkh-2 software and parallelized LKH-2 software using SPC[3] PM for the large size TSP instance calculated for 10 runs

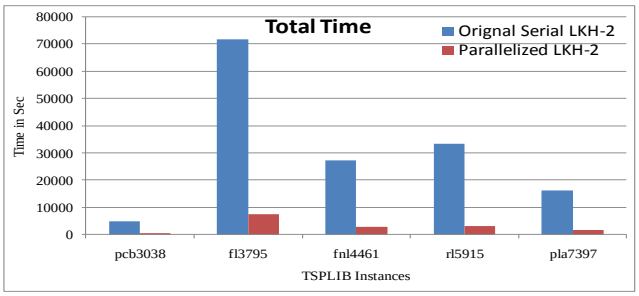

FIGURE 8. Comparison of total execution time between original serial lkh-2 software and parallelized LKH-2 software using SPC[3] PM for the large size TSP instance calculated for 10 runs

From Fig. 6, 7 and 8, same observation can be made for large size TSP instance as that of the medium size TSP instance. The minimum, average and total execution time for parallelized LKH-2 software using SPC[3] PM is found lesser than that of the original LKH-2 software requires.

VII. CONCLUSION AND FUTURE WORK

The results from this study show that the SPC[3] PM (**S**erial, **P**arallel, and **C**oncurrent **C**ore to **C**ore **P**rogramming **M**odel) provides a simpler, effective and scalable way to parallelize a given code and make it suitable for multi-core processors. With the concurrent and parallel function of SPC[3] PM, the programmer can transform a given serial code into parallel and concurrent executable form for making most of the multi-core processors.

The Lin-Kernighan Heuristic (LKH-2) for Solving Travelling Salesman Problem which is generally considered to be one of the most effective methods for generating optimal or near-optimal solutions for the symmetric traveling salesman problem is made further effective and less time consuming by introducing parallelism and concurrency in the algorithm with the help of SPC[3] PM. Besides, the new parallel and concurrent implementation of the algorithm founds much more scalable and suitable for multi-core processors.

This SPC[3] PM will be further worked out for introduction of some more parallel and concurrent functionality and synchronizing tools and will be applied to other standard and classical problems to meet the software challenges of multi-core era.

REFERENCES

[1] M. A Ismail, S.H. Mirza, T. Altaf, “Concurrent matrix multiplication on multi-core processors”, Intl. J. of Comp. Sc. & Security, vol. 5(4), 2011, pp 208-220.

[2] N. Vachharajani, Y. Zhang and T. Jablin, "Revisiting the sequential programming model for the multicore era", IEEE MICRO, Jan - Feb 2008.

[3] M. D. McCool, "Scalable programming models for massively multicore processors", Proceedings of the IEEE, vol. 96(5), 2008.

[4] F. Glover, G.A. Kochenberger, Handbook of Metaheuristics, Kluwer’s international series, 2003, pp. 475-514.

[5] D. L. Applegate, R. Bixby, V. Chvatal, W. J. Cook, The Travelling Salesman Problem, Princeton University Press, 2006, pp. 29, 59-78, 103, 425-469, 489-524.

[6] E. Alba, Parallel Metaheuristics a new class of algorithms, Willey, 2006.

[7] K. Helsgaun, “General *k*-opt submoves for the Lin–Kernighan TSP heuristic”, Math. Prog. Comp., vol. 1, pp. 119–163, 2009.

[8] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.): The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, New York , 1985.

[9] S. Lin, , B. W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem”. Oper.Res. vol. 21, pp. 498–516, 1973.

[10] K. Helsgaun, “An effective implementation of the Lin–Kernighan traveling salesman heuristic” , EJOR 12, pp. 106–130, 2000.

[11] H.H. Hoos, T. Stützle, Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, Menlo Park , 2004.

[12] D. S. Johnson, “Local optimization and the traveling salesman problem”, LNCS, vol. 442, pp. 446–461, 1990.

[13] http://www.akira.ruc.dk/~keld/research/LKH/

[14] http://www.tsp.gatech.edu/data/index.html

[15] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

AUTHORS PROFILE

Muhammad Ali Ismail received his B.E degree in Computer and Information Systems from NED University of Engineering and Technology, Pakistan in 2004. He did his M.Engg in Computer Engineering with specialization in Computer Systems and Design from the same university in 2007. Now he is pursuing his PhD research in the field of Multi-core Processors. His areas of interest include serial, parallel and distributed computer architectures, Shared memory, distributed memory and GPU programming, including generic and specific models and algorithms. He has received first prize Gold medal in all Pakistan competition for his cluster design. He is a member of IEEE (USA), IET (UK) & PEC.

Prof. Dr. Shahid Hafeez Mirza has 38 years of experience in teaching and research. He received his bachelor degree in Electrical Engineering from NED University of Engineering and Technology, Pakistan. He did his MS from USA. From UK he pursued his degree of doctorate. He has served department of Computer and Information System Engineering of NED University as a Chairman. He also remained the DEAN of faculty of Electrical and Computer Engineering. Now he is Senior Research fellow in the same university. His fields of interest include processor design, computer architecture and parallel processing. He is a member IEEE (USA) and member IET (UK).

Prof. Dr. Talat Altaf has 29 years of experience in teaching and research. He has B.Sc. Engineering (Honours) in Electrical Engineering and M.Sc. Engineering (Instrumentation) degrees from Aligarh Muslim University, Aligarh in 1976 and 1983 respectively. He pursued his Ph.D. degree from the University of Bradford, U.K. during 1990 till 1994. He has served the Department of Electrical Engineering of NED University as a Chairman during 1997 till 2007. He is presently the Dean of the Faculty of Electrical and Computer Engineering. His fields of interests are in the areas of Current Mode Circuits/Filters, Digital Signal Processing, Parallel Processing, Energy Conversion and Distributed Generation. He is a member of IEEE (USA), Circuits and Systems Society and Instrumentation and Measurement Society. He is also member of IET (U.K.), PEC, and Illumination Society of Pakistan.