CHAPTER I

INTRODUCTION

1.1 MOTIVATION OF THE WORK

Parallelism in computations can be achieved in a variety of ways. Several algorithms designed on sequential computers, have parallelism transparent in their structures. Parallel version of such algorithms can easily be designed by exploiting their parallelism. However, it is often necessary to restructure a sequential algorithm in order to design a corresponding algorithm for a parallel computer, henceforth referred to as a parallel algorithm. The restructuring involves reorganisation of computation in order to distribute it among the available processors. The reorganisation depends on the architecture of the parallel computer.

Following FLYNN's [69] classification scheme, parallel computers are broadly classified into SIMD (Single Instruction Stream Multiple Data Stream) computers and MIMD (Mutliple Instruction Stream Multiple Data Stream) computers. STONE [214] has presented an overview on parallel computer architecture. In SIMD computers a stream of instructions is executed by a number of synchronized processors, each operating upon its own memory. Examples of SIMD computers are the Burroughs ILLIAC IV, ICLDAP, CDCSTAR-100, Cyber 203, 205, CRI Cray-1, TISAC, Burroughs BSP and PEPE (See [111]). In MIMD computers each processor has an independent instruction counter and operates in a speed independent manner on shared memories. Examples of MIMD computers are Burroughs 'Wind Tunnel Proposal', Carnegie-Mellon Cm* , Cmmp, DYNELCORHEP, Goodyear MPP, Lawrence Livemore Laboratory S1, NASA Langley Finite Element Machine, NOSC/UNIVAC Multiple AP-120s and University of Texas TRAC (See [111]).

In order to use parallel computers most effectively it is necessary to design efficient parallel algorithms. Parallel algorithms have been studied since early sixties (See [143]) although no parallel computer had been built at that time. Interest in designing parallel algorithms has increased after the arrival of parallel computers in computer field. From an algorithmic point of view, SIMD computers have been studied most widely.

One of the primary goals in designing sequential algorithms for solving a problem is to minimize the execution time. Likewise, the execution time of a parallel algorithm is considered as one of the primary measures of the performance of an algorithm. An obvious advantage of using parallel computers rather than sequential computers is the reduction in the execution time for solving a problem. In a sequential algorithm there exists a trade-off relation between the number of operations performed and the storage required by the algorithm. These two factors are considered as the measure of complexity of a sequential algorithm. Since the execution time of a parallel algorithm is directly related to the number of operations performed by each processor, the number of processors used and the storage required by the algorithm, so it is realistic to consider these three factors to evaluate the performance of a parallel algorithm. There exists a trade-off relation in a parallel algorithm among these three factors. These factors are considered as the measure of complexity of a parallel algorithm.

Several numerical and non-numerical problems have been considered for designing parallel algorithms. However, only for a few semi-numerical problems parallel algorithms are available. The thesis is concernd specifically with the design and analysis of parallel algorithms for some semi-numerical and non-numerical problems. The problems considered in the thesis are some generating problems, a selection problem and a sampling problem. The SIMD computer has been chosen as the model of computation to design parallel algorithms. The importance of designing parallel algorithms and the existence of very few parallel algorithms for the problems considered in the thesis have motivated this investigation.

Generating problems occur in combinatorial mathematics. Problems for the generation of sequences such as combinations, permutations with or without repetitions and derangements have been considered for investigation. A good number of sequential algorithms for each of these problems are available. There exists only one parallel algorithm for a generating problem. MOR and FRAENKEL [149] have proposed a parallel algorithm for generating permutations on a vector processor. Absence of any other work in this area is the main reason for designing parallel algorithms for generating combinations, permutations and derangements.

JOHNSON and MIZOGUCHI [109] have proposed a sequential algorithm for the problem of selecting the k-th element of a cartesian sum of two sets of elements. This problem occurs in statistics and oprations research [145]. This type of problem also arises in some VLSI layout problems [145]. On critical examination of JOHNSON and MIZOGUCHI's algorithm it has been found that the algorithm can further be improved for sequential computers. Moreover, parallelism can be exploited efficiently to design a parallel algorithm for this problem. The scope for improvement of the existing sequential algorithm and the nonavailability of a parallel algorithm are the basic reasons for proposing algorithms for selecting the k-th element of a cartesian sum of two sets of elements.

The problem of drawing a random sample without replacement occurs in statistical experiments, quality control, data processing and game problems. The problem may also occur in computer simulation studies where sampling may be repeated many times. According to the conventional definition of combinations of n elements out of N elements, the sampling problem can be considered as the generation of a random combination. Again, the problem can be treated as the generation of the first n elements of a ranom permutation of N elements. Several sequential algorithms for selecting a random sample without replacement are available [62, 83, 114] but there does not exist any parallel sampling algorithm. The interest for designing a fast and storage economical sampling algorithm and the non-availability of parallel algorithms for drawing a random sample are the motivation of designing some algorithms on both sequential and parallel computers.

1.2 AN OVERVIEW OF THE THESIS

The thesis demonstrates specially how SIMD computers can be used effectively in some semi-numerical and non-numerical problems. A sequential algorithm for balanced tree search and insertion is proposed. Parallel algorithms for generating permutations, combinations, permutations with repetitions and derangements on SIMD computers are developed. Each of these algorithms generates the sequences in lexicographic order. A parallel algorithm for a selection problem is designed on a Shared Memory Model (SMM) of an SIMD computer. Efficient sequential algorithms for drawing a random sample are proposed. Some parallel algorithms for drawing a random sample using SMM are also developed.

The thesis consists of twelve chapters. Chapter I, the present chapter, is introductory. It introduces the problems under investigation, explains the motivation of the work and summarises the results.

Chapter II gives brief surveys of the available literature on relative problems and existing parallel algorithms on SIMD computers. The architecture of different models of SIMD computers such as Shared Memory Model (SMM), Mesh Connected Computer (MCC), Cube Connected Computer (CCC) and Perfect Shuffle Computer (PSC) have been described in Chapter III. This chapter presents the conventions used for stating sequential and parallel algorithms. It also states the assumptions on the basis of which an algorithm is analysed.

Chapter IV deals with two problems, one on sequential computers and the other on SMM of SIMD computers. Algorithms presented in this chapter have been used in later chapters. The first problem is a ranking and insertion problem. Given a table of records which form a balanced tree and an argument k, the algorithm determines a non-negative integer u, modifies k by k + u and inserts into the tree a node containing k such that the rank of the inserted node when records are arranged in increasing order of their keys is u + 1 and rebalances the tree if necessary to make the tree balanced. This problem is similar to the balanced tree search and insertion problem considered by KNUTH [113]. The algorithm proposed for this problem has logarithmic time complexity on sequential computers. The second problem considered in this chapter is to compute the cumulative sum of elements of a linear or a two dimensional array using SMM of SIMD computers. More specifically, given a linear array X = {X(i), i = 1,2..., n } of n elements or a two dimensional array M = {M(i,j), j = 1,2...,m and i = 1,2..., t } of t x m elements, the problem is to replace

X(i) by X(1)+X(2)+...+X(i) or to replace M(i,j) by M(i,1)+M(i,2)+...+M(i,j) for all i and j. The algorithms have the logarithmic time complexities.

Chapter V considers the problem of generating permutations on SIMD computers. This chapter presents a parallel algorithm for generating permutations of r elements out of n distinct elements in lexicographic order. The algorithm can be used in any model of SIMD computers. The algorithm requires $O({}^nP_r\ r \log r/P)$ units of time to generate all the distinct permutations where P is the available number of processors. The algorithm has the space complexity of $O({}^nP_r\ r)$ in the common memory when the SMM is considered and of $O({}^nP_r\ r/P)$ in the local memory of each processor when any other model of SIMD computers is used. An example has been considered to illustrate the algorithm.

Chapter VI deals with the problem of generating combinations on an SMM of SIMD computers. A parallel algorithm which generates all distinct combinations of r elements out of n elements in lexicographic order has been presented in this chapter; the elements are assumed to be distinct. The $({}^nC_r)$ combinations contain all possible arrangements made up of elements such that they differ in composition but not in the order of the elements. The algorithm is further improved for $r > n-r$. The algorithm has the time complexity of $O(\min({}^nC_r\ r \log (n-r+1)/P,\ {}^nC_r\ (n-r) \log (r+1)/P))$ and the storage complexity of $O({}^nC_r n)$ in the common memory where P is the number of processors available. An example has been considered to illustrate the algorithm.

Chapter VII considers the problem of generating permutations of n elements where elements may not be distinct. The problem can be stated as follows: given n elements of which r_1 elments are of the first kind, r_2 elements of the second kind, ..., and r_k of the k-th kind so that $r_1 + r_2 + \dots + r_k = n$, it is required to generate all the N_o permutations of n elements lexicographically using parallel computers. Two parallel algorithms have been presented. The first agorithm can be used in any model of SIMD computers. The algorithm has the time complexity of $O(N_o\ n\ k/P)$ when P processors are available. The algorithm requires $O(N_o n)$ storage in the common memory when the SMM is used while it requires $O(N_o$

n/P) storage in the local memory of each processor when any other model of SIMD computers is considered. An example is considered to illustrate the algorithm. The algorithm can further be improved when an SMM is available. Improvements have been considered in order to design a second parallel algorithm in this chapter. The algorithm requires $O(N_o n \log k/M)$ units of time and $O(N_o n)$ storage locations in the common memory when M.k processors are available in the SMM, M being a positive integer and k the number of different kinds of elements. The two parallel algorithms have been compared with the algorithms for generating permutations and combinations in lexicographic order which are proposed in Chapter V and Chapter VI respectively.

Chapter VIII deals with the derangement problem. The problem is to generate the permutations of n elements with certain restrictions. The problem can be stated more specifically as follows: given n distinct elements it is required to generate all permutations in parallel, of n elements lexicographically with the restriction that the i-th element does not occur in the i-th position. A parallel algorithm for generating all distinct derangements of n elements is proposed in this chapter. The algorithm can be used in any model of SIMD computers. The algorithm requires $O(D_n n \log n/P)$ units of time where D_n is the total number of distinct derangements of n elements and P is the available number of processors. The algorithm has the space complexity of $O(D_n.n)$ in the common memory when the SMM is considered and of $O(D_n.n/P)$ in the local memory of each processor when any other model of SIMD computers is considered.

Chapter IX considers a selection problem. Given two linear arrays $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ of n elements each, the problem is to select the k-th element of $X+Y = \{ x_i+y_j \mid x_i \in X \text{ and } y_j \in Y, i,j = 1,2...,n \}$ using an SMM of SIMD computers. The problem has initially been considered for sequential computers by JOHNSON and MIZOGUCHI [109]. Certain improvements on JOHNSON and MIZOGUCHI's approach have been suggested in this chapter. An algorithm for selecting the k-th element of X+Y on an SMM has also been proposed. The algorithm has the time complexity of $O(\log k. \log n)$ and the space complexity of $O(n)$ when $O(n^{1+1/\beta})$ processors are used, β being a preasigned constant lying between Ø and 1.

Chapter X deals with the problem of sampling. Given a population of size N, the problem is to select n sample units for a random sample without replacement. This chapter presents two sequential algorithms for this problem. The basic approach of **Algorithm SELECT** [83] has been used in designing the first algorithm. **Algorithm SELECT** has the time complexity of O(N) and the space complexity of O(N). Following the principle of **Algorithm SELECT** some more algorithms [56, 219] are developed. The best known algorithm is due to TEUHOLA and NEVALAINEN [219] and has the time complexity of O(n^2) in the worst case and O(n) in the average case; the algorithm has the space complexity of O(n). The first algorithm presented in this chapter has the time complexity of O(n log n) in the worst case and the space complexity of O(n). Using the technique of balanced tree search and insertion presented in Chapter IV, a new algorithm for drawing a random sample of size n without replacement has been proposed in this chapter. The algorithm requires O(n log n) units of time and O(n) storage locations. Examples have been considered to illustrate the algorithms.

Chapter XI presents five parallel algorithms for selecting a random sample without replacement. Among the five parallel algorithms, three algorithms can be used even if the population size is not known in advance. The basic principles of **Algorithm TRYAGAIN** [83], **Algorithm RESERVOIR** [114], **Algorithm BINOMIAL** [62], **Algorithm LEAPFROG** [62] and **Algorithm SELECT** [83] have been used in designing the five parallel algorithms. The algorithm designed on the basis of **Algorithm SELECT** is the best among the five parallel algorithms. This algorithm has the time complexity of O(n) and the space complexity of O(n). Each algorithm is illustrated with an example.

The last chapter makes some concluding remarks on the present work. It also poses some problems for further investigation.