CHAPTER I: Introduction

1.1 Motivation

Parallel computing has become one of the major areas of research in the computer science field with numerous journals and conferences dedicated to its development. Multiprocessor computers (parallel computers) are becoming more prevalent, with several manufacturers and models readily available, several at quite low prices. However, while the hardware is progressing rapidly, the methodologies and algorithms necessary to exploit their power are not as well developed. Expanding the use of parallel computers is severely handicapped by the lack of available software. Thus, parallel algorithm design is very important to the success of parallel architectures and parallel computing in general.

The goal of a parallel algorithm is to be able to solve a single problem using several processors working together and to do so in an efficient manner. With respect to parallel computing, it is desirable that the parallel algorithm run faster than its sequential counterpart in proportion to the number of processors used. Such efficient algorithms are termed work-optimal. For many problems, efficient parallel algorithms have been developed, and several design techniques have evolved in the process. However, there are still many problems, particularly in the domain of non-numeric computations, for which work-optimal solutions are yet to be devised. And for some problems, existing strategies do not seem to be effective in leading to efficient parallel solutions.

Tied to the development of parallel algorithms is the issue of the associated parallel data structures. Although many of the general strategies used in sequential processing are easily applied within the parallel environment (e.g., divide-and-conquer), many of the data structures commonly used in sequential computing (queue, stack, linked list) are linear in nature and do not parallelize easily. Therefore, there is a need for new access methods and data structures to be associated with the new algorithm design techniques in order to make the transition to parallel machines.

It is the goal of this dissertation to address these underdeveloped areas. With the expansion of parallel computing into so many areas of application, there is a need to formalize strategies to be used in solving problems. In particular, there is a need to be able to categorize problems and define strategies to more easily solve problems with similar structure and requirements. The intent of this work is to establish a new strategy for designing work-optimal parallel algorithms and to do so for a broad class of problems. The work begins with the study of a well-established strategy for parallel algorithm design, lined list ranking, and then moves to the development of parentheses matching as a new strategy.

1.2 Performance Metrics

To measure the effectiveness of a parallel algorithm, it is necessary to devise a method to measure the performance of that parallel algorithm with respect to a sequential algorithm or another parallel algorithm which solves the same problem. Clearly the goal of a parallel algorithm is to solve a problem faster than its sequential counterpart. Thus, an important measure of the algorithm is the running time. While the actual running time is of concern, this measurement is difficult at best. In addition to the fact that not all models are available to researchers, the dependence of performance upon the actual hardware and the manner in which that hardware is utilized, require that general and easy to compute measures be utilized.

For a given problem let $T_i$ and $T_p$ represent the running times of the best known sequential algorithm and the parallel algorithm using $p$ processors, respectively. Then the speedup of the parallel algorithm is given by $S = T_i / T_p$ and the work is $W = p \cdot T_p$. If $W = O(T_i)$ then the algorithm is said to be work-optimal.
and it achieves *linear speedup* within a constant factor. (The term *cost* is also used in the literature to mean *work*.)

Another desirable characteristic of a parallel algorithm is that of *scalability*. An algorithm is said to be *scalable* if the performance increases linearly with the number of processors utilized. This implies that the algorithm sustains good (expected) performance for a wide range of processors used. (Obviously, there is a point at which adding more processors will no longer reduce the running time.) Therefore, scalability increases the algorithm's flexibility for use on machines with varying numbers of processors.

1.3 Models of Computation

There are numerous parallel models available, some realizable, some theoretical. In general there are two categories of machines - *shared memory* and *fixed connection* topologies. This research incorporates examples of both categories. With respect to the shared memory model, the *parallel random access machine (PRAM)* is emphasized. With respect to the fixed connection models, the *hypercube* model is considered.

A survey of the field of parallel algorithms demonstrates that the preferred and widely accepted model for the design of parallel algorithms is the shared memory *parallel random access machine (PRAM)* model. Although not currently realizable, it provides a well-defined, easy-to-use platform for parallel algorithm design, with algorithms already available for a broad spectrum of problems [3,61,52]. Unfortunately for owners of commercially available, fixed connection parallel computers, the repertoire of algorithms for these machines is not so well developed. However, one popular, commercially available fixed connection computer, the hypercube model has probably generated more activity in this area than any of the others [92,74].

The initial research presented here begins with the development of algorithms for shared memory computers, but subsequent research reflects the interest in the underdeveloped area of fixed connection models, with particular emphasis on the hypercube. This approach will demonstrate the applicability of the results to both ends of the parallel computer spectrum.

1.3.1 PRAM Model

The *parallel random access machine (PRAM)* model consists of \( p \) processors, each directly connected to a shared global memory. All communication among processors is through this shared memory. Each of the \( p \) processors is a general purpose processor having a local memory. In general, all processors may access the shared, global memory simultaneously, with various restrictions applied to different models. Figure 1.1 shows the structure of the PRAM model.

With respect to reading, *concurrent read* indicates that two or more processors may read the same memory location simultaneously; similarly, *concurrent write* indicates that two or more processors may write to the same address simultaneously. At the other extreme, *exclusive read or write* implies that simultaneous access to global memory is allowed only if each processor is accessing a unique address. The four models are therefore *EREW* (exclusive read/exclusive write), *CREW* (concurrent read/concurrent write), *ERCW* (exclusive read/concurrent write) and *CRCW* (concurrent read/concurrent write), the most powerful of the four. In addition, variations of CRCW exist which determine the method by which concurrent writes are resolved. For more details see [3].

The shared memory PRAM model is a widely accepted fundamental, theoretical model for parallel algorithm development, the use of which allows for the development of algorithms without the problems of communication delays, independent memories and distribution of data. Many researchers begin their quest for a parallel solution to a problem using this model, as it allows the researcher to concentrate on the traits of the problem itself with few architectural restrictions.
1.3.2 Hypercube Model

The hypercube is a distributed memory, message-passing, parallel computer. A 4-dimensional hypercube is a fixed connection computer having $N = 2^r$ nodes and $r2^r$ edges. The nodes are numbered 0 to $N-1$ in binary form and are connected so that any two nodes whose binary numbers differ in exactly one bit position are connected by an edge. Figure 1.2 shows hypercubes for $N = 2, 4$ and $8$.

The hypercube computing model provides a different set of considerations to be addressed from the PRAM models. First, it is a message passing model, with each processor having a local, not shared memory. Second, because of the configuration of the hardware, communication delays between processors must be taken into account. Clearly, any parallel algorithm developed for a shared memory computer can be 'forced' onto the hypercube model and the algorithm will find a solution to the problem. The drawback is that if the data distribution and communication are not well planned, the time complexity can increase dramatically due to communication overhead. Therefore, issues such as load balancing [62, 111, 96], data distribution [97], and communication [84, 65, 65] must be carefully considered.

1.4 Parallel Algorithm Design Strategies

A paradigm is defined as a general strategy used to aid in the development of the solution to a problem. The establishment of a paradigm first involves identifying a technique and applying it to the solution of a problem. The next step is to define the algorithmic technique and demonstrate that it is applicable in the solutions of a set of related problems. The establishment of such paradigms is important in that they provide basic approaches through which new problems may be solved or through which previous problems may be solved more efficiently.

There are numerous paradigms associated with parallel algorithm development. These include divide-and-conquer [9,36], branch-and-bound [73,72], and dynamic programming [35,104,76], each of which have been extended from their sequential use. Others, such as deterministic coin tossing [25,26], symmetry breaking [53], accelerating cascades [25], tree contraction [87,1], Euler tours [102], linked list ranking [112,24,28, 6, 54], and all nearest smaller values [63], have been developed specifically for application to the development of parallel algorithms. Although this list is not complete, it serves to provide an overview of some of the most widely used techniques in parallel algorithm design. To demonstrate the applicability of these techniques, a brief descript-
is provided for each. A survey of many of these techniques is given in [109].

The divide-and-conquer paradigm [9,36] is the very basis of parallel algorithm design. The idea is to divide a given problem into several smaller subproblems which can be solved independently. The results from each of the subproblems are then combined into the final solution for the original problem. In the parallel environment, each subproblem is assigned to a separate processor, and all subproblems are solved simultaneously. The results are combined using one or more processors. Clearly, any problem which is to be solved on a parallel computer utilizes some form of the divide-and-conquer strategy.

Dynamic programming [35,104, 76] is a divide-and-conquer technique which is applied when the subproblems of the original problem are not independent, that is, when the subproblems share common subproblems. As subproblems are solved, the solutions are saved in a table for use as needed, as opposed to recomputing a solution each time an instance of a subproblem arises. Dynamic programming is often applied to a category of problems referred to as optimization problems in which a maximum or minimum value for the solution is desired.

Branch-and-bound [73,72] is a breadth-first tree processing technique in which the branch of the tree to be traversed next is dependent upon the current value of a bounding function which is computed as the tree is traversed. The value of the bounding function allows some branches of the tree to be pruned (i.e. eliminated from further consideration), thus reducing the search time. The computation of the bounding function is used to lead to the solution of the original problem. The branch-and-bound strategy is often applied to problems in which the solution is determined by searching through a tree, as in a game problem.

Symmetry breaking [53] refers to a technique by which a linked structure such as a linked list is partitioned into several disjoint pieces. One specific symmetry breaking technique, deterministic coin tossing [25,26], uses the binary representation of the index of each element to select nonadjacent elements for processing. Repeated application of the algorithm allows the user to select elements from the list whose distances between them are within a specified range. This technique is widely used among the various linked list ranking algorithms.

Accelerating cascades [25] involves applying two or more different algorithms to a single problem, changing from one algorithm to another when the ratio of the problem size to the number of processors reaches a certain level, referred to as the threshold. This use of multiple algorithms allows for the fine-tuning of an algorithm in order to obtain better performance than any one algorithm can attain when used alone.

Tree contraction [87,1] is a tree processing technique in which the nodes of a tree are removed, and the information contained in the node being removed is combined with the information contained in its parent. In a parallel environment, processors are assigned to independent nodes so that multiple nodes are removed concurrently. When the tree has been reduced to a single node, the root, the solution to the problem is found in that node. A common application of tree contraction is to tree problems in which the value for a given node is dependent upon the values of the nodes in its subtree, such as expression evaluation.

The Euler tour technique [102] has been successfully applied to many tree and graph problems. It involves duplicating each link in a tree (graph) pointing in the opposite direction of the original link, forming an Euler circuit or path through the tree (graph). This essentially allows the tree (graph) to be approached as if it were a linked list and, thus, allows for easier processing. This is the technique most often applied in algorithms to compute the traversals of trees (graphs).

Linked list ranking [112, 24, 28, 6] is a technique whereby the elements of a linked list are assigned a rank which corresponds to the number of elements succeeding (preceding) it in the list. (See chapter 2 for details.) This technique has been applied to a wide range of tree and graph problems. (See table 2.1.)

Given an array of numbers, the all nearest smaller values (ANSV) problem is to determine, for each number which is smaller than \( x \) [13]. The solution of ANSV has been applied successfully to several \( \cdots \) problems, including the depth first search of an interval graph [33] and parentheses matching, line packing, and triangulating a monotone polynomial [63].
A study of the application of these strategies to various problems solved in parallel reveals that it is seldom the case that only one technique is applied to a given problem. One finds that many problems of interest require a combination of techniques in order to achieve an efficient parallel algorithm. These strategies, when used in combination, serve to complement each other and allow for a wide variety of approaches in algorithm design.

In the parallel environment, there are many problems which have not yet been solved in an efficient manner. And in some cases, it seems that the currently known strategies do not provide the necessary functionality to do so. Thus, the search for new and innovative strategies continues. As a result, this dissertation proposes to establish a new strategy - parallel parentheses matching - by demonstrating its applicability to a variety of problems.

1.5 Research Overview

The remainder of this dissertation is organized into six chapters. Chapter 2 defines the related terminology and summarizes the previous work in the area of linked list ranking, including the evolutionary development and applications. This material is taken from "A Comprehensive Survey of Linked List Ranking Algorithms" (125 pages) [54], which has been submitted to ACM Computing Surveys for possible publication. The complete survey provides detailed discussions of all known linked list ranking algorithms for the various models of parallel computation. In addition, examples for each algorithm and comparisons of the algorithms are provided. This survey is the only known complete collection and analysis of existing list ranking algorithms and provides a thorough background study for persons interested in pursuing research in this area.

Chapter 3 presents two new linked list ranking algorithms. This work is presented in "Simple Deterministic and Randomized Algorithms for Linked List Ranking on the EREW PRAM Model" (TR # CRPDC-92-21, UNT, 12 pages) [38], which has been submitted to Parallel Processing Letters for possible publication. The algorithms presented demonstrate an alternative approach to the traditional reduce-rank-expand technique that has been dominant in the past. In addition, they achieve smaller space and time complexity constants than the previously proposed algorithms.

In chapters 4 and 5, parallel parentheses matching is proposed as a general parallel algorithm design strategy, as presented in the paper "Efficient Parallel Algorithms for Tree-Related Problems Using the Parentheses Matching Strategy" (TR # CRPDC-93-11, UNT 25 pages) [40]. Chapter 4 describes the general strategy of parentheses matching and reviews the existing work in this area. Discussed are the relationships between parentheses and tree structures, the equivalence of the Euler tour of trees and parentheses matching, and the relationship between tree contraction and parentheses. In addition, a number of algorithms relating to these topics are developed.

In chapter 5, several new algorithms are presented in which the parentheses matching strategy is applied to a variety of tree related problems. These include algorithms for the nearest enclosing parentheses problem, the heights of all nodes in a tree, the extreme values of the nodes of the subtrees of a tree, and the lowest common ancestor problem. Finally, parentheses matching is applied to solve the problem of globally balancing a binary tree.

Chapter 6 presents a new hypercube algorithm for parentheses matching from the paper titled "A Divide-and-Conquer Hypercube Algorithm for Parentheses Matching" [39] and provides a comparison to two recently proposed parentheses matching algorithms for the hypercube mode. The distinction of this algorithm is the technique used for ensuring the load balance among the processors of the hypercube. A hypercube solution to the nearest enclosing parentheses problem, which utilizes parentheses matching, is also given. Chapter 7 concludes the dissertation with a summary of the work presented here and with suggestions for future research in this area.