CHAPTER 4

PARENTHESES MATCHING AS A STRATEGY
FOR PARALLEL ALGORITHM DESIGN

4.1 Introduction

In the area of parallel algorithm design, there is a need to categorize strategies in order to efficiently solve a broad class of non-numeric problems with similar structures and requirements. Although several strategies or paradigms\(^1\) have been developed for designing parallel algorithms, there are still problems for which existing techniques do not seem to lead to efficient solutions or for which new paradigms are necessary. Therefore, the search for efficient design strategies continues.

In the search for unified strategies for designing efficient parallel algorithms, several problems (particularly, related to graphs and trees) have been encountered which are elegantly solved by applying the solution of the well-known parentheses matching problem as an intermediate step. Other than the natural application of parsing arithmetic expressions and dynamic expression evaluation [52, 12], this approach has been taken for minimum coloring of interval graphs [33], breadth-first traversal of trees [21], sorting a special class of integers [21], approximate bin packing [5], string and dictionary matching [4], and maximal matching of cographs [77]. Thus, there is current interest and promise in this work.

However, for the majority of these seemingly unrelated problems, there is no obvious correlation with parentheses matching, and they come from different domains

\(^1\)By a paradigm, we mean a general strategy which can be used to solve a wide class of problems.
or applications. Furthermore, parentheses matching has so far been applied on an *ad hoc* basis, and to the best of our knowledge, there has been no effort to formalize this strategy. Hence, the motivation for this work.

In this dissertation, *parallel parallel parentheses matching (PPM)* is proposed as a new strategy for designing efficient parallel algorithms. Influenced by the study of the development of linked list ranking as a design strategy, four primary objectives addressed:

1. Identify problems for which parentheses matching has previously been applied in the solution.

2. Describe the parentheses matching technique and its implementations.

3. Identify specific problems which can be efficiently solved using the parentheses matching strategy and develop the associated parallel algorithms.

4. Identify classes of problems for which parentheses matching is likely to be a good design technique.

Given a problem, the approach involves transforming it into an equivalent parentheses matching problem (which is most often non-trivial) and then applying an efficient parentheses matching algorithm in parallel. The final step is to interpret (crucial) the matching information as it applies to the solution of the original problem. Figure 4.1 shows a schematic diagram of this new strategy.

Parentheses matching as a design strategy is introduced in this chapter through a presentation of the existing work. It begins with a discussion of the independent algorithms which have appeared in the literature. This is followed by a detailed
discussion of its relation to tree related problems. This work clearly demonstrates the applicability of PPM to the class of problems related to trees and provides evidence of its probable application to a variety of other types of problems in the future.

4.2 Existing Work with Parentheses Matching

The *parentheses matching* problem is defined as follows. Given a well-formed sequence of parentheses stored in an array, determine the index of the mate of each parenthesis stored in the array. Generally, it is assumed that the input string of parentheses is stored in an array INPUT[1...n]. (Without loss of generality, assume that $n = 2^k$.) The solution to the problem is an array MATCH[1...n] such that MATCH[i] = j (and, likewise, MATCH[j] = i) if and only if the parentheses in locations i and j of INPUT are mates.

The sequential solution to the parentheses matching problem is quite straightforward using a stack. As the input string is read, left parentheses are pushed onto the stack. When a right parentheses is encountered, the stack is popped. This pair of left and right parentheses is a matched pair. Direct implementation of the sequential solution on common parallel models is not feasible because of the contention problems introduced by the sequential nature of the stack. Therefore, most parallel solutions take a different approach.

Several algorithms have been proposed to solve the parentheses matching problem on the PRAM model of computation. Dekel and Sahni [43] proposed an early algorithm for the EREW model in which they implemented the best known sequential algorithm in parallel. It achieves $O(\log^2 n)$ time when using $n$ processors or $O(\log n)$ time when using $n^2/\log n$ processors. Bar-On and Vishkin [12] proposed an optimal CREW PRAM algorithm based on a tree structure in which the parentheses
are partitioned equally among the processors and each processor determines the local matches within its assigned string. Then each processor determines the match of its leftmost left and rightmost right parentheses and uses this information to match the remaining parentheses in the assigned subsequence. The algorithm achieves \(O(\log n)\) time using \(\frac{n}{\log n}\) processors. The algorithm can also be implemented on the EREW model in \(O(\log^2 n)\) time using \(\frac{n}{\log n}\) processors, thus reducing the number of processors used in [43].

The first optimal EREW PRAM algorithm was proposed by Tsang, et al. in [103]. The three phase algorithm logically organizes the processors into a binary tree and partitions the parentheses string among the leaf processors. First the local matches within each partition are computed. Each processor then sends its remaining parentheses information to its parent processor, where the information from the two children are combined. The information consists of a triple \((m, l, r)\) where \(m\) is the number of pairs matched at that level and \(l\) and \(r\) are the number of unmatched left and right parentheses, respectively. The final phase begins when a triple is received by the root processor, at which point a unique identifier may be assigned to each matched pair. The identifier is assigned as the information returns to the leaf processors. By pipelining the operations in the final stage, the algorithm achieves \(O(\log n)\) time using \(O\left(\frac{n}{\log n}\right)\) processors.

In [34] three parentheses matching algorithms are proposed, two for the EREW model and one for the CREW. One EREW algorithm, although not work-optimal, provides a simple divide-and-conquer approach. It requires \(O(\log^2 n)\) time using \(\frac{n}{\log n}\) processors. (Details are provided in chapter 6.)

The CREW algorithm [34] is work-optimal, achieving \(O(\log n)\) time using \(\frac{n}{\log n}\) processors. The input string of parentheses is distributed evenly among the processors, and each processor determines the local matches within its string. Within
the substring of each processor, the leftmost left and rightmost right parentheses are
marked as the representatives of the string and their mates are found. The unmatched
strings within each processor are then encoded to provide the number of consecutive
left (right) parentheses and the ending (beginning) index of the string. Figure 4.2
demonstrates this encoding scheme. The encoded strings are then merged by pairs
in a tree-like fashion. The matching information is coded as a superscript \( < i, j > \).
On a left parentheses, the superscript indicates that its mate is the \( j^{th} \) unmatched
right parentheses to the left of \( ) \). It is defined similarly for a right parentheses, and
each subscript is distinguished to indicate whether the mate is found while travers-
ing up or down the merge tree. At each stage, matched pairs are removed, and the
remaining strings are re-encoded. Once the mates for some parentheses are found,
the information is provided to the children nodes to continue computing mates of the
parentheses in the children substrings.

\[
)7)8)9(10(11)→3)9(102
\]

Figure 4.2: Encoding of Parentheses

Concurrent reads are necessary when distributing the subscript information be-
tween levels of the tree. However, by duplicating unmatched parentheses information
and carrying it through the tree, the algorithm can be implemented on the EREW
model in the same time bounds, but at the cost of additional space.

The EREW work-optimal algorithm proposed in [34] is referred to by its authors
as a privatized, match-and-copy algorithm. It uses an approach similar to that in the
CREW algorithm, as well as matching arrays and a variety of variables, all stored in
the local memory of each processor. The algorithm requires \( O(n_p + \log p) \) time and
space when using \( p \) processors. Optimal EREW algorithms have also been proposed
by several others, including [19, 46, 75, 94].
4.3 Existing Work in the Application of Parentheses Matching

Other than the natural application of parentheses matching to parallel parsing of arithmetic expressions [44, 43, 52], some attempts have recently been made in applying PPM as a subproblem to the solution of other problems in parallel. These include the Euler tour of trees [22], minimum coloring of an interval graph [18], breadth first traversal of general trees and sorting of integers in a restricted class [21], bin packing [5], string and dictionary matching [4], and cograph matching [77].

However, each of these solutions was developed independently of the others. That is, there has been no known effort to characterize problems that lend themselves to the application of this technique. This dissertation presents the first stage of work in this area. This section provides a brief overview of three of these previously solved problems to familiarize the reader with the nature of the technique.

4.3.1 Breadth-first Traversal of a Tree

Chen and Das [21] designed a simple algorithm for the breadth-first traversal of a general tree using PPM. The algorithm determines the arc sequence corresponding to an Euler tour of the tree to compute the level number of each arc. After deleting the leftmost and rightmost arcs on each level, it assigns a left parenthesis to each backward arc and a right parenthesis to each forward arc. Using parentheses matching, it computes the mate (match) of each parentheses (i.e., arc). The match of the rightmost arc in level \( i \) is defined to be the leftmost arc in level \( i + 1 \).

Let \( A = < x, y > \) indicate an arc leaving node \( x \) and entering node \( y \). A matched pair of arcs \( (A, B) \), where \( A = < x, y > \) and \( B = < w, z > \), is interpreted as \( \text{NEXT}(x) = z \). The solution is completed by defining \( \text{NEXT}(x) = y \) if arc \( A \) is the first arc in the arc sequence. The \( \text{NEXT} \) function converts the tree into a linked
list of tree nodes, and performing parallel linked list ranking enumerates the nodes in breadth-first traversal order. The algorithm runs in $O(\log n)$ time using $\frac{n}{\log n}$ processors on EREW PRAM model.

4.3.2 Sorting Integers in a Restricted Class

Chen and Das [21] also used the PPM strategy to sort a sequence consisting of a restricted subclass of integers (RSCI), satisfying the property that any two consecutive elements in the sequence differ in value by at most one. The underlying idea is to construct a tree such that the level numbers of nodes visited while traversing the tree according to its arc sequence correspond to the given sequence of integers in the subclass.

Two different arc sequences are used on the tree – the Euler arc sequence and the complementary Euler (C-Euler) arc sequence – to determine a sequence of well-formed parentheses. The Euler sequence of parentheses is obtained as in the breadth-first traversal algorithm. The C-Euler sequence is obtained from the original Euler tour (without eliminating any arcs) and then assigning each forward (backward) arc a left (right) parenthesis. Each parenthesis (arc) string is then matched to define the NEXT function, which forms the linked list of integers. Finally, linked list ranking is used to sort the integers in the restricted sequence in $O(\log n)$ time, employing $\frac{n}{\log n}$ processors on the EREW PRAM.

4.3.3 Coloring of an Interval Graph

Given a collection of $n$ intervals $I = \{I_i = [a_i, b_i]|a_i \leq b_i\}$, there exists an interval graph $G_I = (V, E)$ with $n$ nodes such that the node set is $V = \{I_i \in I\}$ and the edge set is $E = \{(I_i, I_j)|I_i \cap I_j \neq \emptyset\}$. The problem is to assign colors to the nodes of the interval graph such that the fewest number of colors are used and no two adjacent
nodes have the same color.

A sequential algorithm can be designed as follows for (minimum) coloring of an interval graph using a stack. First, the endpoints are sorted in increasing order. If a beginning endpoint \( a_i \) is encountered, a color is popped off the stack and assigned to the corresponding node. On the other hand, if its endpoint \( b_i \) is encountered, this indicates that all remaining nodes are not adjacent to it. So the associated color is pushed back onto the stack for reuse.

The algorithm given in [19] uses the PPM technique to replace the stack. After sorting the endpoints in ascending order, a left (or right) parentheses is assigned to each beginning (or ending) point, \( a_i \) (or \( b_i \)). The levels of the parentheses are computed and sorted using the algorithm for sorting the restricted class of integers given in the previous section. Multiple linked lists are then built, one for each distinct level. The first element of each list is given a color, which is distributed to the others in that list using a linked list ranking method. The interval graph coloring problem is thus solved in \( O(\log n) \) time using \( n \) processors on the EREW PRAM model.

4.4 Application of Parentheses Matching to Trees

In order to apply PPM to trees and their related problems, let us first define a number of necessary terms and discuss the relationship between a given tree and an equivalent sequence of parentheses. Then several algorithms are discussed which prove the equivalence between the traditional representations of a tree and a parentheses string representation of a tree.

4.4.1 Representations and Traversals of Trees

The term tree is defined to mean a general tree unless otherwise specified. A
rooted general tree is represented by either

(i) leftmost-child\(v\) = (u) and right-sibling\(v\) = (w) relation, or

(ii) parent-of\(v\) = (u) relation.

A rooted binary tree has an additional representation

(iii) right child\(v\) = (u) and left-child\(v\) = (w) relation.

It is shown in [22] that any of three representations can be converted to another in \(O(\frac{n}{p} + \log n)\) time using \(p\) processors on the EREW model. Therefore, any tree algorithm having \(\Omega(\frac{n}{p} + \log n)\) time complexity is independent of the input data structure on this model.

Several traversals are described here which are necessary to the understanding of the algorithms to follow. The preorder and postorder traversals of a general tree are defined analogous to those for binary trees. However, an inorder traversal is one in which a parent node is 'visited' between each of its children. A combination pre/post order traversal is one in which each parent node is visited exactly twice, once before its children are processed and once after all its children are processed. In addition, each leaf node is processed twice, consecutively. The pre/post node sequence obtained from the tree in figure 4.3 is

\[
\begin{array}{cccccccccccc}
  a & b & c & c' & b' & d & e & h & h' & e' & f & f' & g & g' & d' & a'
\end{array}
\]

with the second occurrence of each node distinguished. The inorder node sequence from the same tree is c b a h e d f d g.

Another traversal commonly defined for a tree is the Euler tour, in which each arc is duplicated, but pointing in the opposite direction. That is, for each arc \(A\) leaving node a and entering node b in the tree, arc \(A'\) is added leaving b and entering a. This addition of arcs forms an Euler circuit within the tree. By starting at the root of the tree and following each arc in sequence, the Euler sequence of the nodes is defined.
For the tree in figure 4.3, the Euler node sequence is

\[ a \ b \ c \ b \ a \ d \ e \ h \ e \ d \ f \ d \ g \ d \ a. \]

The Euler sequence of arcs for the tree is

\[ < a, b > \ < b, c > \ < c, b > \ < b, a > \ < a, d > \ < d, e > \ < e, h > \ < h, e > \ < e, d > \ < d, f > \ < f, d > \ < d, g > \ < g, d > \ < d, a >. \]

A more general representation of an Euler tour is the combination of Euler sequence of nodes and Euler sequence of arcs, where each node is embedded between the arcs.

Chen, Das, and Akl [22] presented a unified tree traversal algorithm which is one method used to produce traversals described above. After computing the node sequence corresponding to an Euler tour of a tree, this algorithm defines a NEXT function which describes how to mark the nodes to be retained for the desired inorder, preorder, or postorder traversal from this sequence. A variation of this algorithm constructs the combination pre/post ordering of the nodes. To do this, both the preorder and postorder nodes are marked and the leaf nodes are duplicated. The unified algorithm requires \( O(\frac{n}{p} + \log n) \) time using \( p \) processors on the EREW model.
4.4.2 Tree Representations Related to Parentheses

In addition to the traditional traversals, other representations – a parentheses string and a parentheses string with embedded nodes – have been found to be very useful in the application of the PPM strategy. Depending upon the nature of the problem to be solved, there are two variations of such parentheses string representations of a tree: either associating a pair of parentheses to a node (node-associated parentheses string) or associating a parenthesis to an edge (edge-associated parentheses string). The primary difference between the two is that the node-associated parentheses string has an additional outermost pair of parentheses. By way of comparison to the Euler tour representations [102], the node-associated parentheses string is comparable to Euler sequence of nodes, and the edge-associated parentheses string is comparable to Euler sequence of arcs.

A more general form of parentheses string representation of a tree is a parentheses string with embedded nodes. This corresponds to associating a left (right) parenthesis to a forward (backward) arc in the Euler tour augmented with nodes. When a tree is given as input, the Euler tour can be constructed so, the node identifiers between two consecutive parentheses are immediately available. Figure 4.4 gives the parentheses representations of the tree from figure 4.3. Shown in (α) and (β) are the node-associated and edge-associated parentheses strings. In (γ) is shown the parentheses string with embedded nodes.
4.4.3 Euler Tour Technique vs. Parentheses Matching

Many tree related problems are easily solved through Euler tour techniques. The basic Euler tour technique consists of three phases: the construction of an Euler tour, the assignment of proper weights to a node or an arc, and parallel linked list ranking. When applying PPM to tree related problems in which the input is not given as a tree but as a string of balanced parentheses, the differences and similarities between these two techniques become clearer. A major difference between the two techniques is that the tree structure is explicit in the input of the tree related problems, but in the parentheses related problems the tree structure is underlying, that is, the parent-child relationships in the underlying tree are computed as needed using parentheses matching and possibly linked list ranking. In terms of computation costs for solving tree related problems, the Euler tour technique constructs the Euler tour first and then applies linked list ranking. On the other hand, parentheses matching technique uses parentheses matching and possibly linked list ranking, depending on the problem. However, the ability of both to solve many of the same problems indicates their similarity.

4.5 Parentheses Matching Applied to Basic Tree Operations

In addition to the applications discussed in section 4.3, Das, et al. [40] recently proposed several useful PPM algorithms dealing with the relationships between trees and parentheses. These include an algorithm which accepts as input a general tree and produces the inorder traversal of the tree. Another algorithm accepts as input a well-formed parentheses string and produces the corresponding tree, with the nodes numbered in preorder form. A third algorithm reconstructs a tree from the preorder and postorder traversals. Each of these algorithms utilizes PPM to accomplish the
given task in $O(\log n)$ time utilizing $O\left(\frac{n}{\log n}\right)$ processors on the EREW PRAM model. In the following section these algorithms are discussed as they serve to establish the direct relationship between parentheses matching and trees.

4.5.1 Tree Traversals via PPM

Depth first traversals (preorder, postorder, and inorder) and the corresponding ordering of the nodes is applicable to a wide variety of tree (and graph) related problems. Application of parentheses matching allows for the development of a relatively simple algorithm. (Preorder and postorder traversals can be solved similarly.)

The algorithm given in [40] first computes the Euler arc sequence of the tree and assigns a left (right) parentheses to each forward (backward) arc. After matching the parentheses, nodes are marked in inorder fashion. That is, each leaf node is determined by a left parenthesis which is immediately followed by a right parentheses. An inner node is determined by a right parenthesis immediately followed by a left parentheses. Finally, the occurrence of a node having a single child is determined when two consecutive right parentheses also have consecutive mates. Each of these nodes is marked by a 1. Performing the prefix sums operation on the list of ones numbers the nodes according to the inorder sequence. The algorithm performs in $O(\log n)$ time utilizing $O\left(\frac{n}{\log n}\right)$ processors on the EREW PRAM model.

Figure 4.5 demonstrates Algorithm INORDER, showing the parentheses with embedded nodes for the tree given in Figure 4.3. Performing the prefix sums on the nodes marked with 1’s completes the inorder numbering of the nodes, as shown in the third row of the table.

4.5.2 Converting Parentheses to Trees

An algorithm for converting a well-formed sequence of parentheses into the corre-
Parentheses with Embedded Nodes

<table>
<thead>
<tr>
<th></th>
<th>a ( b ( c ) b )</th>
<th>a ( d ( e ( h )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>e ) d ( f ) d ( g ) d</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 4.5: Demonstration of Algorithm INORDER

sponding tree form in which each node has a pointer to its parent used with the nodes numbered in preorder form is also given in [40]. Because this algorithm is utilized in several application algorithms (in chapter 5) in order to convert the parenthesis string back into the desired tree, it is reproduced here. The idea is to select representative nodes from the dummy nodes embedded in the parentheses string according to the desired node numbering. Broadcasting the computed node numbers to the rest of the dummy nodes and checking the nodes surrounding each left parenthesis suffices to identify parent nodes. The algorithm constructs the tree in \( O(\log n) \) time using \( O(\frac{n}{\log n}) \) processors on the EREW model. Constructing a tree where the nodes are numbered in postorder can be computed similarly.

Algorithm PARENTHESES-TO-TREE [40]

*Input*: a string of parentheses, \( A[1..n] \).

*Output*: an ordered, rooted tree with nodes numbered in preorder, \( PARENT[1..\frac{n}{2} + 1] \).

1. Perform Parentheses Matching on \( A[1..2n] \): store results in \( MATE[1..2n] \).
2. Identify representative nodes in preorder numbering.
For $1 \leq i \leq 2n$ pardo

if $A[i] = \text{'}(' \text{'} then $HEAD[i + 1] := 1$

$HEAD[1] := 1$

Compute the prefix-sums of $HEAD$, store in $NUM$.

3. Connect duplicate dummy nodes representing the same tree node.

For $1 \leq i \leq 2n$ pardo

if $A[i] = \text{'}(' \text{'} then $NEXT[i - 1] := MATE[i] + 1$

4. For $1 \leq i \leq 2n$ pardo

if $HEAD[i] = 1$ then broadcast $NUM[i]$ along $NEXT$ pointers, storing results in $NUM$

(i.e. $NUM[NEXT[i] := NUM[i]]$

5. Determine parents.

For $1 \leq i \leq 2n$ pardo

if $A[i] = \text{'}(' \text{'} then $PARENT[i + 1] := NUM[i - 1]$

Example

The example in figure 4.6 demonstrates the algorithm for the tree in figure 4.3. The embedded parentheses string is given in row A. The values in $MATE$ are found using parallel parentheses matching. Each node identifier immediately following a left parenthesis is marked with a one in the $HEAD$ field, as is location one. Then the prefix sums are computed on the $MARK$ values and the results are stored in $NUM$. ($NUM$ also provides the mapping of node numbers to node identifiers as listed in A.) For every node immediately preceeding a left parenthesis, its $NEXT$ value is set to be one greater than the $MATE$ value of the succeeding left parenthesis. This forms a linked list of the duplicate instances of each node identifier. For each node identifier
Figure 4.6: Demonstration of Algorithm PARENTHESSES-TO-TREE

having HEAD = 1, the corresponding NUM value is broadcast along the linked list. This organizes the NUM values so that for each left parenthesis, the NUM value to its left is the parent of the NUM value on its right, thus allowing for the creation of the PARENT array.

4.5.3 Constructing a Tree from Traversals

Given the preorder and postorder node sequences of a general tree the problem is to reconstruct the unique binary tree. The algorithm given in [40] compares the relative ordering of nodes in the two traversals to determine parent-child relationships, linking two nodes if the order of two consecutive elements in the two traversals is reversed, (indicating a parent-child relationship). Linking the parent and child causes the creation of several disjoint linked lists, which, when interconnected form a linked list corresponding to the tree to which the parentheses are assigned. After performing
parentheses matching, the tree is generated using Algorithm PARENTHESES-TO-TREE. The tree is reconstructed from its traversals on the EREW model in $O(\log n)$ time using $\frac{n}{\log n}$ processors. A discussion of the correctness of the algorithm is found in [40].

Using this technique the parent is connected to leftmost child in the preorder sequence, the leftmost child is connected to the parent in the postorder traversal, identical leaves in both traversals are connected, and adjacent siblings are connected. The insertion of two parentheses, i.e., ‘)’ ‘(’, between siblings, constructs the parentheses version of the Euler tour of the tree. Thus the algorithm constructs the unique tree corresponding to the given preorder and postorder traversals. However, the algorithm works correctly only for a general tree in the sense that when a node has a single child it is always considered the leftmost child.

4.5.4 Tree Contraction by Parentheses Matching

It appears that there may be a direct relationship between tree contraction and parentheses matching in that many tree problems solved using PPM can also be solved using the tree contraction technique. The tree contraction problem is the problem of systematically reducing a rooted tree to its root. Abrahamson et al. [1] give an elegant algorithm for this problem which is based on binarization [30] of the input tree and removing odd numbered leaves and their parents at odd iteration and even numbered leaves and their parents at even iteration. The conflicts at parent nodes are avoided by left-right alternation within each iteration. The key to this algorithm is converting a general tree of $n$ nodes into a binary tree of $n$ leaves and $n - 1$ internal nodes, and removing the leaves and corresponding parent nodes by odd-even, left-right alternation. Thus, it reduces the number of leaves and internal nodes by half at each iteration, achieving $O(\log n)$ time and $O(n)$ work.
An algorithm that accomplishes tree contraction using the PPM strategy is given in [40]. This algorithm differs from that in [1] in that (i) it does not require an explicit tree data structure, (ii) the binarization is applied not to a tree but to a string of parentheses, and (iii) it uses only parentheses matching and parallel prefix. This algorithm is useful when the input has a natural representation in a linear array but has an underlying tree structure, as in the evaluation of an arithmetic expression. It also provides insight into the nature of the tree contraction problem and demonstrates the versatility of the proposed parallel parentheses matching strategy. Using the optimal parentheses matching and parallel prefix sums algorithms, Algorithm TPCM is work-optimal, requiring $O(n + \log n)$ time using $p$ processors on the EREW PRAM model. Because the tree contraction algorithm is such a widely used technique, the algorithm from [40] is reproduced here.

**Algorithm TPCM** [40]

*Input:* a rooted general tree, $T$ ($|V_T| = n$)

*Output:* contracted tree node, i.e., a single node.

1. Construct Euler tour from the tree.
2. Construct a string of parentheses by assigning a left parenthesis to a forward arc and a right parenthesis to a backward arc, stored in array $A[1..2n + 1]$. (All even locations are for parentheses and odd locations for nodes.) {If the input is given in the form of a string of parentheses, skip steps 1 and 2.}
3. Match parentheses.
4. Binarize the parentheses.
   a) Compute NEXT pointers, connecting the duplicate nodes in the string.
      
      If $A[i + 1] = \text{'}$ then $NEXT[i] \leftarrow Mate[i + 1] + 1$
   b) Assign weights where new parentheses are to be inserted.
      
      {We assume that there is a dummy \text{'} \text{'} after the end of array $A$.}
then WEIGHT[i] ← 2
else WEIGHT[i] ← 0.

c) Compute prefix-sums of the weights along the Next pointers by using multiple linked list ranking, storing the results in WSUM[i].

d) Adjust weights to correctly reflect the displacement.
   Reinitialize WEIGHT[i] to 1 for all i
   if A[i - 1] = (') and A[i + 1] = (' then WEIGHT[i] ← 3
   if A[i - 1] = (') and A[i + 1] = (' then WEIGHT[i] ← 3 and
   WEIGHT[i + 1] ← WSUM[i] + 1


e) Compute prefix-sums of the weights along the consecutive indices of array A, storing the results in LOC[i].

f) Copy the nodes and parentheses in array A into array A' according to the new location, LOC[i].

g) Insert new parentheses.
   if A'[i - 1] = (') then A'[i + 1] ← ('
   if i = odd and A'[i - 1] = ('(empty) then A'[i + 1] ← (')

h) Apply parentheses matching to the binarized parentheses.

5. Tree contraction by parentheses removal.

   a) Number the terminal parentheses pairs from left to right. (Assign number from 0)

   {The 0th terminal parentheses pair and the last one are not removed throughout step b) as in typical tree contraction.}

   b) for \[\lceil \log_2(n - 2) \rceil + 1 \text{ times do} \]

      (1) Remove odd numbered left terminal parentheses pair.
(2) Remove nearest enclosing parentheses pair of the parentheses pair removed in step (1).

(3) Remove odd numbered right terminal parentheses pair.

(4) Remove nearest enclosing parentheses pair of the parentheses pair removed in step (3).

(5) Renumeral the remaining terminal parentheses pairs by simply halving the terminal ordering number.

{When removing a pair of parentheses, we are updating the left and right parent pointer using the result of parentheses matching in step 4.h.}

C) Remove the last two remaining pairs of parentheses.

{The terminal parentheses pair number 0 and the last one are removed.}

Table 4.1 demonstrates Algorithm TCPM for the tree given in Figure 4.3. A is the original parentheses string, A' is the binarized parentheses, row x.ol (or x.or) shows the parentheses after the removal of odd-left (or odd-right) terminal parentheses pair and its nearest enclosing parentheses pair, and row 4 is the (empty) parentheses, equivalent to a single contracted node at the end.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.ol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.ol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.ol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.6 Conclusion

This chapter has presented the initial stages of the development of parentheses matching as a design strategy. An overview of the existing work which uses PPM and
several implementations of PPM have been given. This existing work in parentheses matching, though much of it has been conceived independently of the others, when brought together demonstrates the validity of the development of parentheses matching as a general strategy. The application of parentheses matching to trees in [21] and to other related structures such as interval graphs [33] and cographs [77] supports the validity of that work. The fact that parentheses matching has also been applied to a variety of seemingly unrelated problems such as sorting integers [21], approximate bin packing [5], and string and dictionary matching [4] serves to emphasize its promise for use in the future.

The establishment of the relationships between tree representations and parentheses strings and between Euler tours and parentheses matching is well-developed in [40]. The relationship between parentheses matching and tree contraction is also addressed [40]. This leads to the work presented in the following chapter which proposes several algorithms which apply parentheses matching to a variety of tree related problems, further supporting the establishment of PPM as a general algorithm design strategy.