15-492: Parallel Algorithms **Lecturer:** Guy Blelloch
Topic: Trees II **Date:** September 25th
Scribe: Kanat Tangwongsan

9.1 Lecture Outline

- Rake-compress tree contraction.
- Applications of tree contraction: arithmetic expressions, line breaking

Remarks: Trees are nice. We can sometimes get more parallelism simply by representing our problem as a tree.

9.2 Analyzing Parallel Tree Contraction

We will analyze the following variant of the tree-contraction algorithm of Miller and Reif.

- **Input:** a binary tree rooted at r (in general we only need bounded degree for the bounds to hold, but we will analyze the binary case for simplicity).
- **Output:** a single node.

The algorithm performs a sequence of contraction steps, each consisting of a rake operation and a compress operation (in any order). The rake operation removes all the leaf nodes in parallel. The compress operation finds an independent set[1] of unary nodes and splice out the selected nodes.

The following lemma implies that we will finish in $O(\log n)$ rounds in expectation.

Lemma 9.2.1 *The number of nodes of after a contraction step is reduced by a constant factor in expectation.*

Before proving this lemma, we will introduce a few notations and prove a claim about binary trees. Given a binary tree T, we can partition the nodes of T into 3 groups: T_0 contains all leaf nodes (out-degree 0), T_1 (unary) contains all nodes with 1 child (out-degree 1), and T_2 (binary) contains all nodes with 2 children. Clearly we have $V(T) = T_0 \cup T_1 \cup T_2$. We relate the number of leaf nodes and binary nodes as follows.

Claim 9.2.2 $|T_0| = |T_2| + 1$.

Proof: The proof is by a strong induction on the number of nodes. It is easy to see that the base case of $n = 1$ trivially holds. Now assume the claim holds for any tree with at most n nodes and let a tree T with $n + 1$ nodes be given. Let r be the root of T. There are two cases:

[1] A set I is an independent set on the graph $G = (V, E)$ if and only if for all v_1, v_2, $(v_1, v_2) \notin E$.

- If r has two subtrees, we define the following quantities. For the left subtree of r, let $T_0^{(L)}$ and $T_2^{(L)}$ denote the leaf nodes and binary nodes, respectively. Let $T_0^{(R)}$ and $T_2^{(R)}$ be defined similarly for the right subtree. By the IH, we know that $|T_0^{(L)}| = |T_2^{(L)}| + 1$ and $|T_0^{(R)}| = |T_2^{(R)}| + 1$. Furthermore, we know that T has $|T_0^{(L)}| + |T_0^{(R)}|$ leaf nodes and $|T_2^{(L)}| + |T_2^{(R)}| + 1$ binary nodes, so

$$\underbrace{|T_0^{(L)}| + |T_0^{(R)}|}_{\#\text{ of leaf nodes of } T} = |T_2^{(L)}| + 1 + |T_2^{(R)}| + 1 = \underbrace{(|T_2^{(L)}| + |T_2^{(R)}| + 1)}_{\#\text{ of binary nodes of } T} + 1,$$

  proving the claim.

- If r has only one subtree, consider the subtree of r. We know that the subtree has the same number of binary nodes and the same number of leaf nodes. It follows directly from the IH that the claim holds for T since the root is a unary node.

■

We now prove the lemma, which follows almost immediately from the claim and a simple observation.

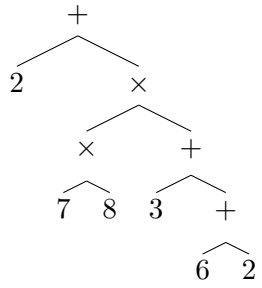
Proof of Lemma 9.2.1: Let m denote the number of nodes before a contraction step, and let m' denote the number of nodes after it. We will show that $\mathbf{E}[m'] \leq 3m/4$. Note that in a contraction step, the rake operation drops all T_0, and the compress operation removes at least $|T_1|/4$ nodes in expectation. The contraction step keeps all of T_2. Therefore,

$$\mathbf{E}\big[m'\big] \leq |T_2| + \frac{3|T_1|}{4} \leq \frac{3}{4} + \frac{3|T_1|}{4} + \frac{3|T_2|}{2} = \frac{3}{4}\left(1 + |T_1| + 2|T_2|\right) = \frac{3}{4}(|T_0| + |T_1| + |T_2|) = \frac{3m}{4}.$$

Note that the second to last equality follows from the claim above. ■

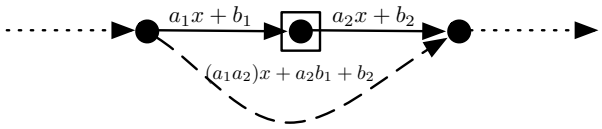
9.3 Arithmetic Expressions

We now apply tree contraction to the problem of evaluating arithmetic expressions. Given an arithmetic expression (e.g., $(3 + 5) * 2 + 7$), we want to evaluate it to a real number (in this case, 23). We assume for simplicity that we work with the real numbers, and our expressions only concern $+$ and $\times$. It is not hard to see that an arithmetic expression has a natural tree representation. For example, the expression $2 + ((7 \times 8) \cdot (3 + (6 + 2)))$ corresponds to the following tree:

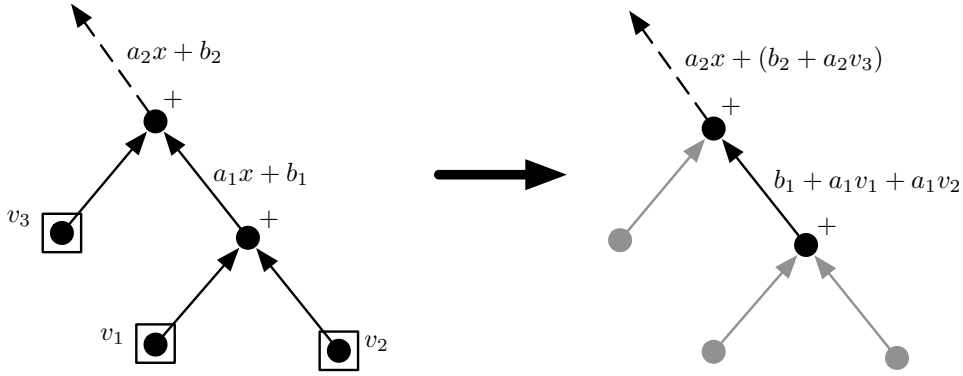


which evaluates to 618. This example shows that an expression tree can be skewed. We will see how tree contraction can help evaluate such a tree in $O(\log n)$ rounds.

Before applying tree contraction, we need to answer the following questions: when we rake a child, how do we incorporate the value at the child node to the parent? Also what do we do when we compress a path? It turns out that if we store the function of the form $ax + b$ on each edge, we can support the rake and compress operations easily. As an example, consider the following situation: compress is getting rid of the middle node (arrows are pointing in the direction of the root node).



As another example, consider the effects of the rake operation on the following tree.



There are multiple ways to do rake. This example here shows just one possibility.

9.4 Line Breaking

Consider the following problem:

- **Input:** an array A of word lengths, and a line length ℓ.
- **Output:** list of words on each line.

For example, for the input $\ell = 12$ and $A = [5, 7, 3, 9, 2, 11, 4, 6, 3, 2]$, we would output [[5, 7], [3, 9], [2], [11], [4, 6], [3, 2]]. At first glance this problem does not seem to have a tree structure in it at all. It is an interesting question to see if one can use the scan primitive to solve this problem.

Here is a sketch of a parallel algorithm that runs in $O(n)$ work and $O(\log n)$ depth. First we compute two vectors from A: L is the the plus-scan on A, and L' is obtained by adding ℓ to every element of L. For our running example, the arrays L and L' look as follows.

$$L(\texttt{plus\_scan}) = [0, 5, 12, 15, 24, 26, 37, 41, 47, 50]$$
$$L' = [12, 17, 24, 27, 36, 38, 49, 53, 59, 62]$$

Then, for each element e of L', we compute the position e would be in L. This step be can accomplished in $O(n)$ work and $O(\log n)$ depth using a variant of a merging algorithm. This step defines the relation: "if I start a line, you should start the next line." The graph given by this relation is a tree, so we can use a leaffix computation (with binary operator `or`) to determine the leaf to root path that contains words which appear at the beginning of a line.