15-750: Graduate Algorithms April 12, 2016

Lecture 31: Parallel Expression Evaluation, Parallel Tree Contraction

Lecturer: Gary Miller *Scribe: Zhong Zhou*

1 Motivation

Depth-First Search and Breadth-First Search are useful for sequential tree algorithms but not quite in parallel algorithms. In the parallel algorithms, there are two approaches commonly [1]:

1. Top-Down approach: Divide-and-Conquer Algorithm
2. Bottom-Up approach: Parallel Tree Contraction

The main difficulty with Divide-and-Conquer is finding the point of seperation, which itself may take $O(\log n)$ [1]. In this lecture we focus on the latter Bottom-Up approach. And in this lecture, Parallel Expression Evaluation serves as an appetizer for Parallel Tree Contraction. And we will introduce both in details in the following sections.

2 Parallel Expression Evaluation

We would like to evaluate an binary expression tree where each node represents either a binary operator or a numerical value in parallel. Figure 1 is an example of expression tree. We are interested in the parallel evaluation.

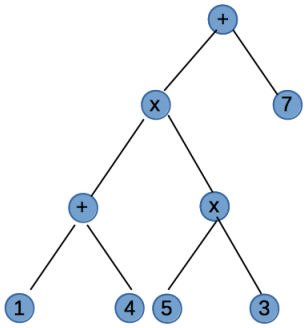

Figure 1: Example of an Expression Tree

Let us consider the simple algorithm below:
Input: T, an expression tree
Output: r, result of evaluation
Assign a processor to each node;
while T *is non-empty* **do**
  if *leaf* **then**
    Send value to parent;
    delete node [RAKE];
  end
  if *node has two children containing values* **then**
    Evaluate the children using the operator;
  end
end

Algorithm 1: Simple Algorithm for Parallel Expression Evaluation

The worst case for the above algorithm is shown in Horner's rule which is shown in Figure 2. This will take $O(n)$ Time, and $O(n^2)$ Work.

Definition 2.1 (Horner's Rule)**.** A rule for polynomial computation that is encapsulated as $a_0 + a_1x + a_2x^2 + ... + a_nx^n = a_0 + x(a_1 + x(... + x(a_{n-1} + x(a_n))...)$.

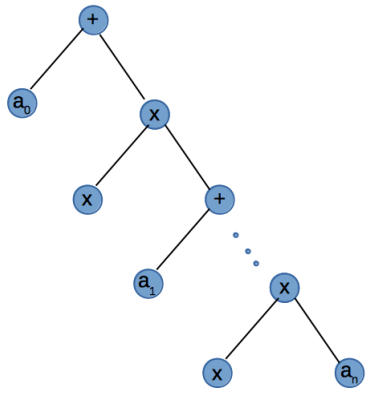

Figure 2: Horner's Rule as an Expression Tree

The intuition for improving upon the naive algorithm is that we would like to keep nodes with 1 missing value busy and we would like to do parallel processing on a large independent set. Firstly, Figure 3 shows us three cases of how to keep nodes with 1 missing value busy. The general case is shown by 4 where $f(y) = ay + b, g(y) = cy + d$, and $f(g(y)) = a(cy + d) + b = (ac)y + (ad + b)$. The linearity is preserved in composition.

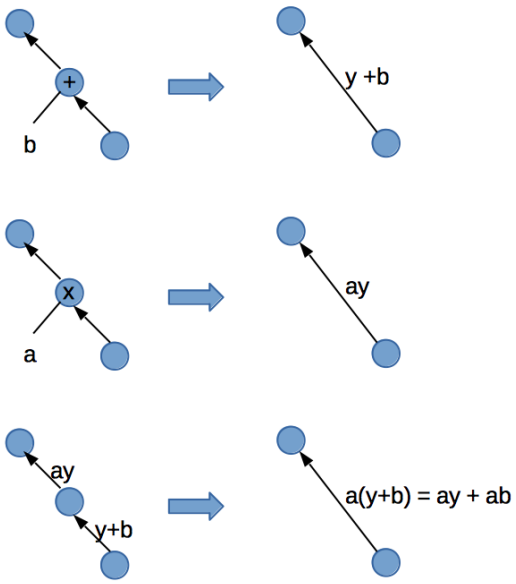

Figure 3: Three Cases of Composition at Nodes with 1 missing Value

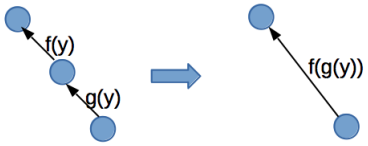

Figure 4: General Case of Composition

Now we consider parallel processing on the independent set:

1. all leaves
2. max independent set from each maximal chain

Definition 2.2 (Chain)**.** $v_0, v_1, ..., v_k$ is a chain if :

1. v_{i+1} is the only child of $v_i, 0 \leq i < k$
2. v_k has only one child and it is not a leaf

Figure 5 is an example of how we will proceed with our new parallel algorithm.

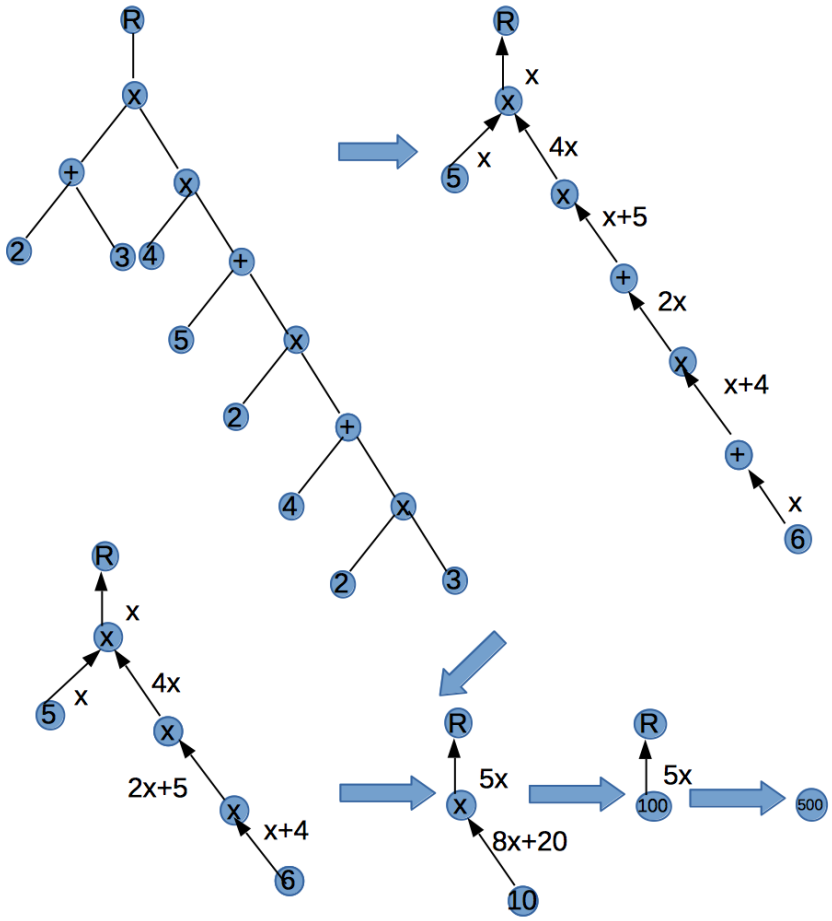

Figure 5: An Example of Evaluation

3 Parallel Tree Contractions

The RAKE idea is also important for Parallel Tree Contraction. In addition to it, we need another operation COMPRESS. The operation CONTRACT will encapsulate both RAKE and COMPRESS and will apply to the entire tree at the same time.

1. RAKE = remove all leaves
2. COMPRESS = replace each max chain of length k with one that is of length $\lfloor k/2 \rfloor$
3. CONTRACT = {RAKE,COMPRESS}

Theorem 3.1. $|CONTRACT(Tree)| \leq 2/3|Tree|$

Proof. Note that CONTRACT is composed of both RAKE and COMPRESS, we can consider each. At the higher level, note that RAKE removes all leaves and COMPRESS replaces each maximal chain with a new chain of length $\lfloor k/2 \rfloor$, we would like to show that both effectively shrink the tree size at each iteration.

Let us define V_0 = leaves of T, $V_1 \subset V$ with 1 child , $V_2 \subset V$ at least 2 children , $C \subset V_1$ with child inV_0. Then the RAKE set is $Ra = V_0 \cup V_2 \cup C$ and the COMPRESS set is $Com = V_1 - Ra$.

Claim 3.2. $|V_0| > |V_2|$

Proof. We can prove the above statement either by induction on the depth of the tree or the size of the tree. We will present the latter here.

We will prove by induction on number of nodes in the tree. For the base case, it is trivially true when the tree is a node, $|V_0| = 1 > |V_2| = 0$. Suppose $|V_0| > |V_2|$ for all trees of size $N-1, N \geq 2$, then for any tree T of size N, let L be a leaf of the tree of size N, let P be the parent of L. Removing L, we will have a tree of size of $N-1$ which fulfils $|V_0| > |V_2|$ by induction. Let V_0^k and V_2^k denotes the set of leaves and set of nodes with two children for tree size k where $k \in \mathrm{Z}^+$.

Now suppose P has only one child, L, then P becomes a leaf upon the removal of L, therefore, $|V_0^N| = |V_0^{N-1}| - 1 + 1 = |V_0^{N-1}| > |V_2^{N-1}| = |V_2^N|$. Suppose P has two child, then P becomes a parent of one child upon the removal of L, then $|V_0^N| = |V_0^{N-1}| + 1 > |V_2^{N-1}| + 1 = |V_2^N|$. Therefore, in both cases, we have $|V_0| > |V_2|$ for trees of size $N, N \geq 2$ for induction phase.

Therefore, we have $|V_0| > |V_2|$.

□

Claim 3.3. $|V_0| \geq |C|$

Proof. $C \subseteq V_1$ with child in V_0, therefore it is trivially true that for each element in C, there will be one element (its child) in V_0. Therefore, it is trivially true that $|V_0| \geq |C|$. □

Having shown these two results, let us consider the RAKE and COMPRESS operations.

$$RAKE(Ra) \subseteq V_2 \cup C \leq 2/3|Ra|, Ra = V_0 \cup V_2 \cup C$$

Since COMPRESS replaces each max chain of length k with one that is of length $\lfloor k/2 \rfloor$:

$$|COMPRESS(Com)| \leq 1/2|Com|, Com = V_1 - Ra$$

Therefore, $|CONTRACT(Tree)| \leq 2/3|Tree|$. □

Using the result, we have the corollary below.

Corollary 3.4. *After* $\log_{3/2} n$ *CONTRACTs, the tree is empty.*

4 Application of Parallel Tree Contraction

We will look at three simple applications of Parallel Tree Contraction.

4.1 Max Value in Subtree

We consider rooted binary tree T with node weights. Our goal is to compute the max value in the subtree for each node.

Our idea is to use Parrallel Tree Contraction here, and we need to define what to do at the RAKE and COMPRESS stage. At the CONTRACT stage, we always take the maximum of the current node and its parent's values. Note that we have removed nodes at the CONTRACT stage (more specifically the COMPRESS operation in CONTRACT) and we need to fill in the values of the removed nodes at the EXPAND stage. Therefore the EXPAND stage focuses on recovering the value of the removed nodes at CONTRACT stage (more specifically, the COMPRESS operation). In order to recover the value of the missing node, we take the maximum value of the node with its child.

We will later show max value in ancestor tree example which is the exact dual of our problem. We will make use of parallel tree contraction as shown below:

Operation	CONTRACT	EXPAND
RAKE	$V(P) = \max\{V(P), V(L)\}$	$\emptyset$
COMPRESS	$V(P) = \max\{V(N), V(P)\}$	$V(N) = \max\{V(N), V(C)\}$

In the RAKE stage, for every leaf L, we denote its parent as P. In the COMPRESS stage, For every node N, we denote its parent as P and its child as C.

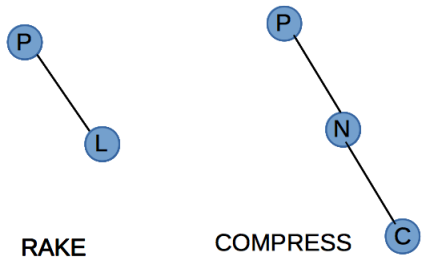

Figure 6: Max Value example

4.2 Max Value in Ancestor Tree

This is the dual of the previous problem on finding max value of subtree. And we just need to modify our parallel tree contraction matrix as the following:

Operation	CONTRACT	EXPAND
RAKE	$\emptyset$	$V(L) = \max\{V(L), V(P)\}$
COMPRESS	$V(C) = \max\{V(C), V(N)\}$	$V(N) = \max\{V(N), V(P)\}$

In the RAKE stage, for every leaf L, we denote its parent as P. In the COMPRESS stage, For every node N, we denote its parent as P and its child as C. Please refer to Figure 6.

Notice that the CONTRACT and EXPAND are exactly reversed when we compare Max Value in Ancestor Tree with Max Value in Subtree.

4.3 Lowest Common Ancestor

Lowest Common Ancestor is a very rich problem that has many good applications in Bioinformatics. For example in phylogenetic tree ("the tree of life") which is the tree connecting all species in creation in a hierarchical fashion, we would like to find the nearest common parent of any two given species, say, dolphin and elephant. We will need to find the lowest common ancestor between dolphin and elephant in the phylogentic tree.

Formally, input of a rooted tree T and non-tree edges $e_1, ..., e_m$, and we would like to find lowest common ancestor $LCA(v, w)$ for each edge $e_i = (v, w)$. In Figure 7, $LCA(b, e) = a, LCA(c, e) = c, LCA(d, d) = d$.

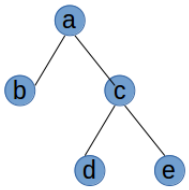

Figure 7: Example of Lowest Common Ancestor

With parallel tree contraction, we have an algorithm for finding the lowest common ancestor for each pair of vertices in the edge set. Whenever the pair is arranged such that one is the ancestor of another, the ancestor is returned as the LCA, if the arrangement is otherwise, we will consider the parents of the vertices instead. The entire algorithm is exactly parallel tree contraction.

Input: $T = (V, E)$, $E = \{e_1, e_2, ..., e_m\}$
Output: $LCA(u, v), \forall e_i = (u, v)$
for $e_i = (v, w) \in \{e_1, e_2, ..., e_m\}$ ***in parallel*** **do**
  for $(u, u') \in \{(v, w), (w, v)\}$ ***in parallel*** **do**
    if u *is ancestor of* u' **then**
      $LCA(e_i) = u$;
    else
      $u = parent(u)$;
      $u' = parent(u')$;
    end
  end
end

Algorithm 2: An Algorithm for Lowest Common Ancestor

5 Reference

[1] Miller, Margaret R. Miller, Gary L. "List Ranking and Parallel Tree Contraction." Synthesis of Parallel Algorithms: 115-194.