CS 598: Communication Cost Analysis of Algorithms
Lecture 16: Tree contraction, Euler tour, list ranking, connectivity, and MST

Edgar Solomonik

University of Illinois at Urbana-Champaign

October 17, 2016
Parallel prefix (scan)

Before continuing with tree contraction, let's consider a simpler problem:

- **parallel prefix**: given array $v \in \mathbb{R}^n$, compute $P(v) = w \in \mathbb{R}^n$, so $w(i) = \sum_{j=1}^{i-1} v(j)$

- compute $z = P(y)$ recursively where $y \in \mathbb{R}^{n/2}$ and $y(i) = v(2i) + v(2i+1)$

- then obtain $w(2i) = z(i-1)$, $w(2i+1) = z(i-1) + v(2i)$ where $z(0) = 0$

- can compute with $O(\log(n))$ steps and $n$ processors in PRAM

Q: how many steps if we use $n/\log_2(n)$ processors?

A: $O(\log(n))$, for recursive step $i$ need $\max(1, \log_2(n)/2^i)$ steps, total less than $3 \log_2(n)$
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.
Tree contraction: expression evaluation ex.

\[ 5888 + 23 \times x \]

986
Tree contraction: expression evaluation ex.
Deterministic rake-compress

For a binary tree, raking leaves can be done in $O(1)$ steps

- consider larger branch factors for a boolean expression tree

Q: if each node computes $\lor$ or $\land$, how can we rake in 1 CRCW PRAM step?

A: if $\lor$, write 1 for all children marked 1, if $\land$, write 0 for all children marked 0 (any conflict resolution is correct)

- rake can be done deterministically, by splitting each chain

worst case: chain of length $n$, completes in $O(\log(n))$ steps but $\Theta(n)$ nodes require work at each step
Randomized parallel compress

Randomization enables a compress step that actually removes nodes
- randomly assign 1 or 0 to each node in the chain
- pointer chase from every node marked 0 whose parent is marked 1

- each rake-compress step decreases the number of nodes by $7/8$ w.h.p.
- Q: why does this give us an algorithm that requires $O(\log(n))$ steps
  and only $O(n/\log(n))$ processors?
- A: first rake-compress with $O(n/\log(n))$ processors takes $O(\log(n))$
  steps, each subsequent rake-compress requires a factor of $7/8$ fewer
  steps, so total $O(\log(n))$
Randomized Miller and Reif algorithm in BSP

So, how do we do tree contraction in the BSP model?

- perform $O(n)$ accesses and pointer chases needed in a PRAM step using $O(1)$ BSP supersteps and $O(n/P)$ communication
- with each step of rake-compress we decrease the number of nodes and accesses geometrically
- need to assume the nodes/accesses are load balanced (can randomly permute initially)
- the communication cost then goes down geometrically
- after $O(\log(P))$ steps, the size of the tree is $O(n/P)$, so we can collect all nodes on one processor and contract the tree locally
- the total cost is then

$$O(n/P \cdot \beta + \log(P) \cdot \alpha)$$
Indexing elements of a linked list

**List ranking** is closely related to tree contraction

- given a linked list $p$ of size $n$, compute the distance $d_p(i)$ from the end of the list for each element
- more generally, scan on a linked list
  - trivial linear time algorithm sequentially
  - can convert to array via first definition, do scan on array, convert back
  - can also perform scan by contracting list and expanding back
- compress by pointer jumping: e.g. compute $q(i) = p(p(i))$, compute $d_q(i)$ then $d_p(i) = 1 + 2d_q(i)$, $d_p(p(i)) = 2d_q(i)$
- same problem as in compress for tree contraction: how to resolve conflicts?
- again randomized solution is good, assign 0 or 1 to each $i$, pointer chase if 0 and $p(i)$ is 1
  - to get $d_p(i)$ keep track of non-unit neighbor distances while recursing
- same asymptotic cost as rake-compress, easy in EREW
Efficient computation of tree contraction in EREW can be done by decomposing into $n/P$-bridges\(^1\) and contracting each bridge to a vertex.

\(^1\)diagram source and further information: Gazit, Miller, Teng 1988
Euler tour

We can find the bridges via an Euler tour\(^2\)

followed by a list ranking (prefix sum) on the Euler tour tree

\(^2\)diagram source: Wikipedia (David Eppstein)
Cost of tree contraction in other models

Both list ranking and rake-compress are based on pointer-chasing

- how expensive is it to perform $n$ chases with $P$ processors?
  - in PRAM, $n/P$ steps
  - in BSP, 1 superstep, $O(n/P)$ communication
  - in the ideal cache model, with cache line of size $L$, $O(Ln/P)$ memory bandwidth cost (each chase is likely a cache miss)
  - in the $\alpha - \beta$ model, we have all-to-all-v
    - by direct send: $n/P$ communication with min($n/P, P-1$) messages
    - by butterfly all-to-all (if load-balanced) $O(\frac{n}{P} \log(P) \cdot \beta + \log(P) \cdot \alpha)$
    - so list ranking and tree contraction cost a factor of $O(\log(P))$ more than in BSP
    - the amount of work is $O(n/P \cdot \gamma)$, flop-to-byte ratio $O(1/ \log(P))$

- conclusion: pointer chases require lots of messages and random (unstructured) memory accesses

- we can do tree contraction faster, if each processor starts with a subtree (even better, $n/P$-bridge)
<table>
<thead>
<tr>
<th>Tree contraction</th>
<th>Practical considerations</th>
</tr>
</thead>
</table>

Short pause
Finding connected components

Consider finding the connected sets of vertices in a (disconnected) graph
- sequentially, there are many linear-time solutions, including BFS
- in parallel things are much more interesting
- Shiloach and Vishkin (1980) provide an efficient CRCW PRAM algorithm
- given a graph with $n$ vertices and $m$ edges, it uses $n + m$ processors to complete in $O(\log(n))$ steps
Parallel algorithm for connectivity

Start with a tree for each node and compute a tree for each connected component

- let each node $i$ store ‘parent’ $F(i)$
- let a star be any tree of height $\leq 2$

The algorithm iterates the following steps

1. **conditional star hooking**: if $(i, j) \in E$, $i$ in star, and $F(i) > F(j)$, perform $F(F(i)) \leftarrow F(j)$ (for every star, some hook may succeed)

2. **unconditional star hooking**: if $(i, j) \in E$, $i$ in star, and $F(i) \neq F(j)$, perform $F(F(i)) \leftarrow F(j)$ (for every star, some hook succeeds)

3. **shortcutting**: (pointer chasing) if $i$ not in star, $F(i) \leftarrow F(F(i))$

and terminates when all nodes are in a star (no hook occurs)
A graph with two connected components
First iteration

1. conditional star hooking
First iteration

2. unconditional star hooking

Connectivity Shiloach-Vishkin algorithm
First iteration

3. shortcutting
Second iteration

1. conditional star hooking
Second iteration

2. unconditional star hooking
Analysis of parallel tree connectivity

Algorithm converges after $O(\log(n))$ iterations
- sum of tree heights (starts at $n$) decreases by a factor of at least $3/2$ every iteration
  - steps 1 and 2 will hook every star to a tree
  - step 3 will decrease the height of every tree by $3/2$
- requires $O(n + m)$ work per step
- $O(\log(n))$ steps with $O(n + m)$ processors in PRAM
- Q: in BSP, can we do $O(\log(P))$ rather than $O(\log(n))$ steps
- A: not easily, cost proportional to $O(\log(n))$ SpMVs with adjacency matrix, plus pointer chasing
Minimal spanning tree (MST)

Given graph $G$ construct spanning tree with minimal sum of edge weights

- if $G$ not connect, spanning tree forest is desired
- Prim’s algorithm: start a tree from random vertex, connect minimal edge to tree
- Kruskal’s algorithm: start a tree at every vertex, add minimal edge that connects two trees
  - works for finding forests
  - given two connected parts of the spanning tree (incl. single vertex), minimal edge connecting these must be in the spanning tree
- this condition suggests a parallel algorithm
Parallel MST algorithm

We follow the approach Shiloach and Vishkin, which is similar to connectivity

- algorithm works for CRCW PRAM with priorities (processor with smallest index wins write conflict)
- start by sorting edges by weight across processors
- perturb each edge weight to make all different or break ties dynamically
- algorithm consists of similar steps
  1. *unconditional star hooking*: if \((i, j) \in E\), \(i\) in star, and \(F(i) \neq F(j)\), perform \(F(F(i)) \leftarrow F(j)\) (for every star, minimal-weight hook succeeds)
  2. *shortcutting*: (pointer chasing) if \(i\) not in star, \(F(i) \leftarrow F(F(i))\)

- if we don’t have priorities, need \(O(\log(n))\) steps to calculate the minimal-weight hook at every iteration