Carnegie Mellon University

Optimizing Deterministic Work-Efficient Parallel Batch Dynamic Tree Contraction

William Gay & Daniel Anderson *(Carnegie Mellon University)*

Introduction

- A dynamic tree can be used to maintain a vertex disjoint forest and efficiently ($O(\log n)$) answer quires about paths and subtrees
- They also allow one to efficiently update the forest by inserting or removing an edge
- In the parallel batch-dynamic setting, the goal is to concurrently handle batches of k updates
- Rake-Compress (RC) Trees are an implementation of Parallel Batch-Dynamic Trees.
- Randomized RC Trees can handle updates in $O(\log n)$ span w.h.p and $O(k \log(1 + n/k))$ expected work.
- Deterministic RC Trees can handle updates in $O(\log n \log\log n)$ span and $O(k \log(1 + n/k))$ worst-case work.
- Our goal is to optimize the deterministic algorithm to reduce the gap in span between the algorithms

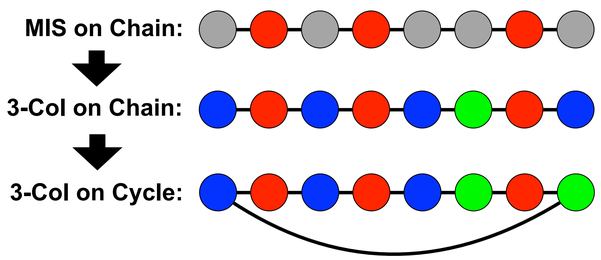
Next Steps

- It is currently unclear whether batch-updates for RC Trees can be achieved deterministically in $O(\log n)$ span while remaining work efficient
- Since an MIS on k vertices can be achieved in $O(\log^\star k)$ span, it seems plausible that one might be able to achieve an update algorithm with $O(\log n \log^\star k)$ span
- An alternative route may be to find an algorithm which doesn't require finding an MIS at all, although this seems difficult because currently, all algorithms for RC Trees require finding an IS
- In either case, the ability to induce direction on the forest provides a promising avenue for future exploration as each vertex now has a unique parent which can be used for symmetry breaking

Contact:
wgay@andrew.cmu.edu

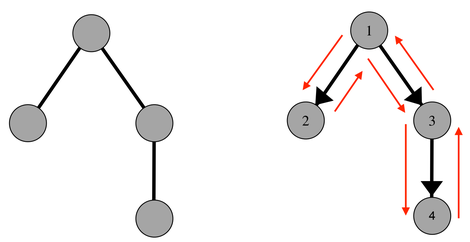
Attempt 1: A Better MIS

- **Deterministic RC Trees Work by iteratively contracting an independent set of vertices of degree 1 & 2**
- This requires finding and sufficiently large independent set of vertices to contract at each iteration
- The current algorithm finds maximal independent set (MIS) , which is guaranteed to be large enough
- There is a constant span reduction from finding an MIS on a chain to 3-coloring a cycle, which has a lower bound of $\log^\star n$ rounds, meaning the current algorithm cannot achieve $o(\log^\star k)$ span per iteration



Attempt 3: Rooting the Tree

- **A rooted tree may provide a way to perform symmetry breaking and allow for a more efficient MIS**
- By maintaining an Euler Tour Tree on the forest maintained by the RC Tree we can arbitrarily induce root in the forest
- An Euler Tour Tree can be deterministically updated in $O(k \log(1 + n/k))$ work and $O(\log n)$ span
- Each vertex in the forest can keep track of its position in the tour, and can efficiently determine it parent by finding its neighbor which appears earliest in the tour.



Inducing direction on a tree via an Euler Tour

Attempt 2.1: Optimizing the Static Algorithm

- **The static algorithm for constructing an RC Tree can be optimized by splitting it into two phases, which each use a different subroutine for to find a MIS to contract**
- The subroutine used in the first phase is work efficient but span inefficient
- The subroutine used in the second phase is span efficient but work inefficient

MIS Subroutine	Work	Span
Subroutine 1	$O(n)$	$O(\log\log n)$
Subroutine 2	$O(n \log^\star n)$	$O(\log^\star n)$

- We use the work-efficient sub-routine for $2\log\log n$ iterations and the span efficient subroutine for $\log(n/\log^2 n)$ iterations
- The result is that the overall algorithm is both work and span efficient

Total Work: $\left(\sum_{k=0}^{2\log\log n} \left(\frac{5}{6}\right)^i O(n)\right) + \log\left(\frac{n}{\log^2 n}\right) O\left(\frac{n \log^\star n}{\log^2 n}\right) = O(n)$

Total Span: $2\log\log n \; O(\log\log n) + \log\left(\frac{n}{\log^2 n}\right) O(\log^\star n) = O(\log n \log^\star n)$

Attempt 2.2: Generalizing the Static Algorithm

- **The batch-update algorithm cannot be optimized in the same way**
- We again consider two different subroutines for finding a MIS

MIS Subroutine	Work	Span
Subroutine 1	$O(k)$	$O(\log\log k)$
Subroutine 2	$O(k \max(c, \log^{(c)} k))$	$O(\log^{(c)} n)$

- Subroutine 1 is work efficient, but span inefficient
- Subroutine 2 can be made span efficient, but work inefficient, by choosing $c = \log^\star n$
- Any choice of c will cause subroutine 2 to be work inefficient, as the choice for c which minimizes $\max(c, \log^{(c)} n)$ is asymptotic asymptotical equivalent to $\log^\star n$
- For certain values of k (e.g. $k = \sqrt{n}$), any way the algorithm gets split into two phases with these subroutines will cause the algorithm to be either work or span inefficient

$$k \approx \sqrt{n} \implies \frac{\log n \log^\star k}{\log\log k} < \log(1 + n/k)$$

- Using subroutine 2 for $\omega(1)$ rounds causes the algorithm to be work inefficient, but using subroutine 1 for $\Omega(\log(1 + n/k))$ rounds causes it to be span inefficient