Parallel algorithm

Core ideas

- **Sequential**: single procedure sweeping all the vertices sequentially.
- **Parallel**: multiple procedures (local growths) running simultaneously.
  - A local growth is started at every minimum.
  - Each local growth spreads independently with an ordered BFS.
  - Each local growth updates its own preimage graph G_r.
  - **Join saddles**: wait until all involved local growths have reached the saddle, then join them.
  - **Split saddles**: the new open edges in $\mathcal{R}(f)$ are handled by the same local growth.

Parallel algorithm

Local growths

Data structures

Each local growth keeps:

- a **Fibonacci heap** θ to store candidates for the ordered BFS;
- an **ST-tree** G_r to store the preimage graph.

candidates are sorted by f value

can be merged in $O(1)$

can be merged in $O(1)$

Join saddles

What if a saddle joins components from different local growths?

- **Detection**: before processing a vertex v, check whether all the vertices in $\mathrm{Link}^-(v)$ have already been visited.
- **Stopping**: if not, terminate this local growth.
- **Processing**: otherwise, this local growth is in charge of proceeding;
  - join the priority queues (θ) and the preimage graphs (G_r) of all local growths terminated at v;
  - process v as usual.

concretely, update an atomic counter visitedLower[v] and check whether visitedLower[v] $= |\mathrm{Link}^-(v)|$

Parallel algorithm

Local growth implementation

```
1  procedure LocalGrowth(v_0, R, Φ)
2      θ, G_r ← {v_0} [Fibonacci heap], ∅ [ST-tree]
3      while θ ≠ ∅ do                    ▸ add |{w ∈ Link⁻(v) : w visited by this local growth}|
4          v ← vertex in θ with minimal f value
5          update visitedLower[v]
6          if visitedLower[v] < |Link⁻(v)| then
7              append (θ, G_r) to pending[v]          } critical section
8              terminate
9          end
10         foreach (θ', G_r') ∈ pending[v] do
11             θ.join(θ'); G_r.join(G_r')
12         end
13         process v, updating G_r, R and Φ
14         add vertices in Link⁺(v) to θ
15     end                               ▸ just as in the sequential algorithm
16 end
```

Parallel algorithm

Full implementation

input : a triangulated mesh $\mathcal{M}$
a scalar field f on $\mathcal{M}$
output : the augmented Reeb graph $(\mathcal{R}, \Phi)$

1 **begin**
2 $\mathcal{R}$, $\Phi \leftarrow \varnothing$ [graph], $\varnothing$ [function]
3 $V \leftarrow$ `FindMinima`$(\mathcal{M}, f)$ → easy to run in parallel
4 **foreach** $v_0 \in V$ **in parallel do**
5 start procedure `LocalGrowth`$(v_0, \mathcal{R}, \Phi)$
6 **end**
7 **return** $(\mathcal{R}, \Phi)$
8 **end**