Algorithm Notes for Merge Tree Computation

Peer-Timo Bremer

1 Concepts

Some random thoughts

- There exist several different *indices* by which we can identify vertices :
  (1) the global order defined by the global mesh
  (2) the block order defined by a regular subset of the data
  (3) the local order defined as compact order of pre-filtered vertices

2 Current Single Core Streaming Algorithm

Input.

- Sequence of *vertices*, v_i, *edges*, e_j, and *finalization info*, f_i.
- Each vertex contains a global index and $k > 0$ coordinates of which the first one is used as function.
- Each edge contains two global indices
- Each finalization info contains one global index

Output.

- Potentially refined *merge tree*
- Segmentation in local order
- Global indices in local lorder

Steps.

1. Parse the data and filter based on a function range. This implicitly creates a new order of vertices. The original vertex ids are stored for later drawing. *Needs and index map from global to local index space to adjust edge indices*
2. Buffer the function value and all attributes that are needed for statistics for all vertices that have passed the filterw. Create minimal vertex token $(id, function)$.

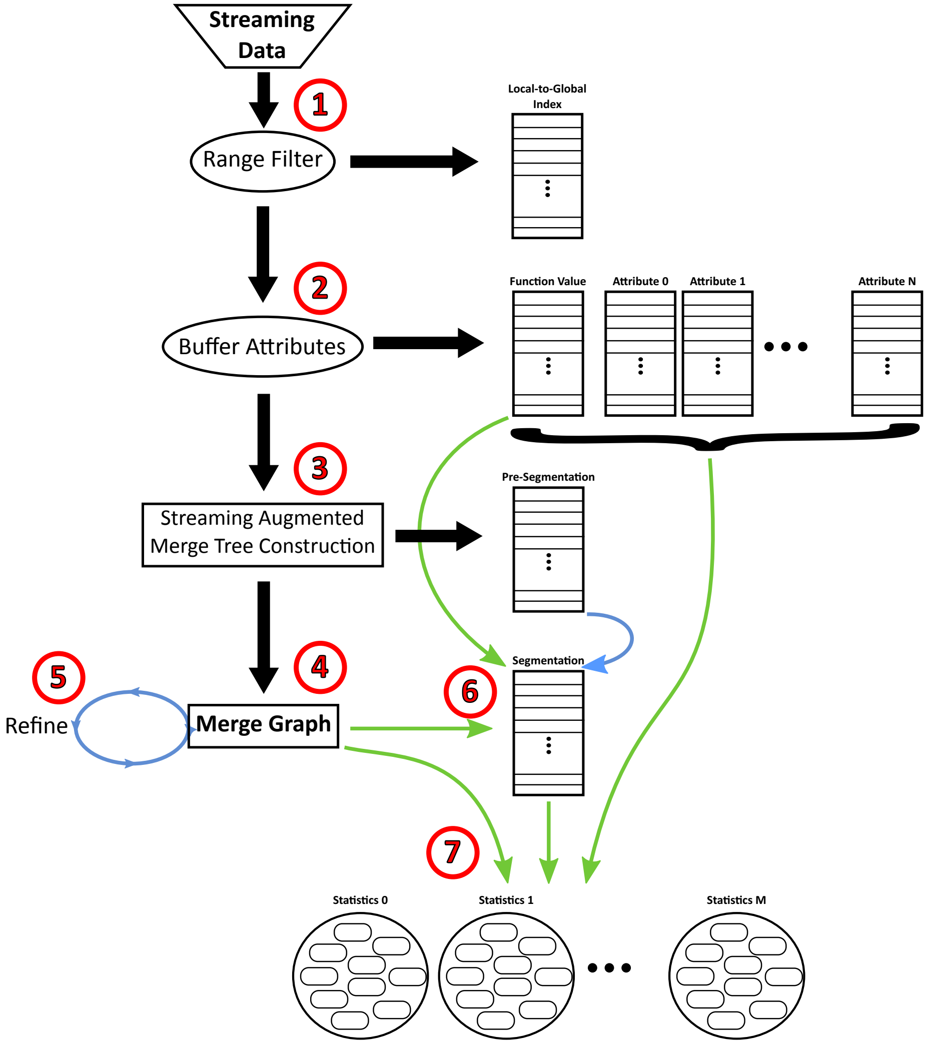

Figure 1: Single core streaming algorithm outline.

3. Streaming construction of the merge tree. Each time a vertex is removed from memory record its last segmentation index.
   Needs the local, compact index space for dense storage

4. Collect the nodes and arcs as they are finalized from the construction algorith.
   Nodes are identified in the local index space.

5. Refine the graph to shorten overly long arcs. This is done in place, New nodes will be marked as not belonging to the orginal mesh (they are *virtual.*

6. Correct the pre-segmentation in place using a path-compression type lookup.
   Each segmentation id corresponds to a vertex id. We must be able to find the corresponding vertex in the array (trivial in the local index space).

7. Collect the attributes of all vertices into sets of statitics one for each feature / node *Needs a map from feature / node id to a compact representation*