WCP → WCT Speedups

Brett Viren and Haiwang Yu

April 3, 2024

Summary

Numbers from memory, we started with the initial job:

- $\approx 90\,\mathrm{s}$ for full job including startup and all I/O.
- $\approx 73\,\mathrm{s}$ for clustering itself.

Major changes (times are from "slow" i7-4770K @ 3.5 GHz):

1. Break full job into imaging and clustering parts.
   - No speed up, just avoid re-running imaging for testing.
2. Redesign the k-d tree interface NFKD $\rightarrow$ `NFKDVec`
   - Reduces clustering to $\approx 11.5\,\mathrm{s}$ about half due to data conversion
3. Fix `is_connected()`
   - Reduces `clustering_live_dead()` from $3.3\,\mathrm{s}$ to $1.9\,\mathrm{s}$.

We also looked config to not sample "extra" point data, saves $\mathcal{O}(1\,\mathrm{GB})$.

- Observe possible over-retention of `shared_ptr`, needs checking.

NFKD → NFKDVec

NFKD's primary motivation: save memory by using blob/node-local PCs in-place.

- `nanoflann` requires we give it a point for a given point "index".

This requires two major features:

1. a "flattening" of cluster-blob-PC-point tree structure to array-like.
2. a "transpose" from N-arrays-of-coordinates to array-of-Ndim-points.

Provided by the iterators of `disjoint_range` and `coordinate_range`.

- With the "scoped view" governing the tree descent.

The problem: `disjoint_range` has **fast sequential access** but **slow random-access** while `nanoflann` requests points **semi-randomly**.

- Root problem: design followed forming a **wrong conclusion** that `nanoflann` loads points **at start** and via **sequential scan**.

The solution: NFKD`Vec` simply copies the disjoint PC coordinate arrays into monolithic vectors to enable fast random-access.

Fix `is_connected()`

We noticed creating `Cluster` facades on "dead" blobs was $10\times$ slower than "live" blobs despite only having $2\times$ more dead than live.

The problem: The `Cluster` constructor and `is_connected()` used `multimap` inefficiently (and wrongly).

- It mapped every "tick" across blob span lead to $\mathcal{O}(1k)$? entries for each "dead" blob.
- Iteration in `is_connected()` re-checked a "live" blob in all ticks of a "dead" blob even after a connection was found.

The result also added the same blob multiple times to output vector.

- We were not able to quickly understand why this did not break subsequent code.

The solution: store blob in `multimap` by only **start time**, in `is_connected()` loop over "dead" blobs first, inner loop visits each "live" blob exactly once, inner loop short-circuits on the "time" check to avoid `overlap_fast()` calls.

Remaining slow points

Component level measured by Pgrapher

2.9 s	`ClusterFileSource`
2.2 s	`PointTreeBuilding` (with no "extra")
10.1 s	`MultiAlgBlobClustering`

Clustering level (`MultiAlgBlobClustering`)

9.0 s	total body of `operator()`.
2.2 s + 1.3 s	convert `ITensor` → PC tree (live+dead)
1.9 s	`clustering_live_dead()`
1.0 s + 0.3 s	convert PC tree → `ITensor` (live+dead)

1 s discrepancy between time measured by `Pgrapher` vs inside `MultiAlgBlobClustering`?

Memory issues

Most important first:

1. `ExecMon` (`MemUsage`) never reports any negative memory delta.
   - This "should" happen when `IData` and facade `shared_ptr`'s are dropped.
2. Curate the point cloud to limit sample point values.
   - Removing the "extra" values saves $\mathcal{O}(1\,\mathrm{GB})$, some "default" values also unneeded?
3. Change `INode` to support eager drop of input data: issue 290.