Scalable Asynchronous Connected Components Detection Library

Senthil Kumar Karthik, Jaemin Choi, University of Illinois Urbana-Champaign

Overview
- Finding connected components: popular graph algorithm used in science and engineering
- A Union-Find based parallel library for distributed memory machines
- Scalable implementation using Charm++
- Performance evaluation on NCSA Blue Waters

Charm++
- Migratable object and task-based parallel programming model
- Adaptive runtime system
- Decompose problem domain into communicating objects (shares)
- Overdecomposition: many more objects than PEs (CPU cores)
- Asynchronous method invocation via messages

Background
- Connected component: a subgraph where vertices are connected by paths, but are not connected to any other vertices outside the subgraph

- Union-Find
  - Operations performed on a disjoint-set data structure
  - Used to detect connected components
  - Union(x,y): merge two sets where vertices x and y belong to each set
  - Find(x): return the unique ID of the set containing x
  - If vertices of interest are in different sets (determined by Find) but the graph says otherwise, merge the sets (Union)

Algorithm
- Adapted version of Shiloach-Vishkin (SV) algorithm
- Perform only tree-hooking step
- Use asynchronous messaging on a distributed graph

For each edge $(v_i, v_j)$ in graph,
1. Message $v_i$ to perform Find$(v_j)$
2. Recursive parent messaging to reach boss
3. Boss messages $v_i$ for Find$(v_j)$
4. Recursive parent messaging to reach boss
5. Set boss, as parent of boss

```
union_request(v_i, v_j) {
  if (v_i.ID > v_j.ID)
    union_request(v_j, v_i)
  else
    find_boss(v_i, v_j)
}

Listing 1: union_request
```

```
find_boss(v_i, v_j) {
  if (v_i.parent == -1)
    find_boss2(v_i, boss)
  else
    find_boss(v_i.parent, v_j)
}

Listing 2: find_boss
```

```
find_boss2(v_i, boss) {
  if (v_i.parent == -1) {
    if (boss > v_i.ID)
      union_request(v_i, boss)
    else
      v_i.parent = boss
  }
  else
    find_boss2(v_i.parent, boss)
}

Listing 3: find_boss2
```

Implementation
- Library involves 3 phases for connected components detection
  - Phase 1: Build forest of inverted trees using asynchronous Union-Find
  - Phase 2: Label each vertex with ID of its boss
  - Phase 3: Prune out insignificant components
- Tested and verified with real-world graphs

Optimizations
- Motivation
  - Highly communication-intensive: lots of tiny messages (~1.5B messages for 16M vertices with 6M edges)
  - Deep trees causing slow Find operations
- Locality-based tree climbing
  - Sequentially traverse tree path for vertices in the same share
  - Increases work per share, but drastically reduces number of messages
  - 25x speedup in tree construction
- Message aggregation
  - Topology-aware routing and aggregation of network communication using TRAM library
- Local path compression
  - Make local tree in each share completely shallow
  - Provides one-hop access to bosses

Probabilistic Mesh
- Random graph built on a lattice structure
- Edge between two lattice points (vertices) determined from a probability value using vertex coordinates
- Easy to scale graph size, verify results and catch race conditions

Future Work
- Integrate with ChaNGa
  - Galaxy detection based on Friends-of-Friends algorithm
  - Detect clusters of stars and classify galaxies