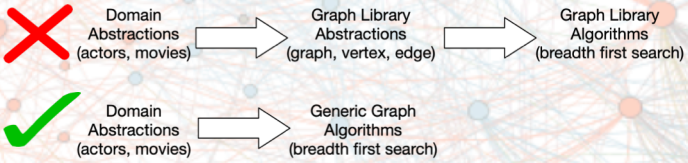
SI2-SSE: GraphPack: Unified Graph Processing with Parallel Boost Graph Library, GraphBLAS and High-Level Generic Algorithm Interfaces

Andrew Lumsdaine (PI), Kevin DeWeese, University of Washington
{al75,deweeskg}@uw.edu

Award 1716828

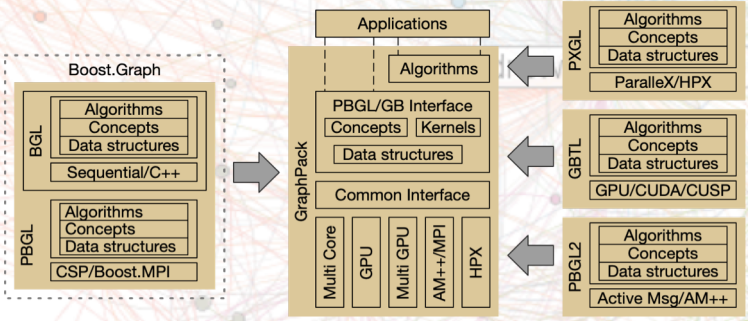
Motivation

- Graphs are a **powerful abstraction** that represent (arbitrary) relationships between (arbitrary) entities
- Graph theory **does not depend on** particular entities or relationships (abstract algorithms)
- Software **does depend on** particular entities and relationships (concrete data structures)
- A **generic** library allows domain abstractions to be used **directly** with library algorithms (no conversion to library types needed)



Architecture

- Build on significant prior work by PI (BGL, PBGL, AM++, GBTL)
- Refactor and modernize for C++17 and beyond



Parallelization

- Parameterized parallelism with C++ **execution policies**
- Sequential, shared memory threads, accelerators
- (Explicit parallelism with C++ tasks)

```
template <typename ExecutionPolicy typename Iterator>
size_t triangle_count(ExecutionPolicy&& policy, Iterator first, Iterator last) {
  atomic<size_t> triangles = 0;
  counting_output_iterator counter(triangles);

  std::for_each(policy, first, last, [&](auto&& x) {
    for (auto v = x.begin(); v != x.end(); ++v) {
      std::set_intersection(v, x.end(), G[*v].begin(), G[*v].end(), counter);
    }
  });

  return triangles;
}
```

Tech Transfer

- Open Source Release April 2020 https://https://gitlab.com/al75/bgl17
- Proposal to ISO C++ Standards Committee SG19: P1709

Concepts / Generic API

- **"There are no graphs"**
- Generic algorithms **specify requirements** on their input types
- Generic libraries contain **useful algorithms** organized around **classifications** of these requirements (**"concepts"**)

```
template<typename A>
concept bool Adjacency() {
  return Random_access_iterator<A>()
  && requires (Value_type<A> adj) {
       typename Inner_iterator<adj>;
       adj.begin() -> Inner_iterator<adj>;
       adj.end() -> Inner_iterator<adj>;
     }
  && Forward_iterator<Inner_iterator<Value_type<adj>>>()
  && Convertible<Inner_iterator<Value_type<Value_type<adj>>>, Difference_type<adj>>();
}
```

- Satisfied, e.g., by `std::vector<std::forward_list<int>>`
- Only need **standard library containers** to use graph algorithms

Example: BFS

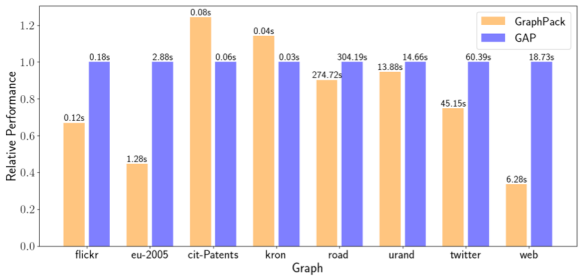
- **Range adapters** allow traversal in specified (algorithmic) order

```
adjacency<0> A { /* initializer... */ };

std::vector<size_t>      distance    (A.size());
std::vector<vertex_id_t> predecessor(A.size());

for (auto&& [u, v] : bfs_edge_range(A)) {
  distance[u]    = distance[v] + 1;
  predecessor[u] = v;
}
```

Algorithms (Current List)

- Breadth First Search
- Depth First Search
- Connected Components
- Triangle Counting
- Bellman-Ford
- Page Rank
- Maximal Independent Set
- Dijkstra
- Edmonds-Karp Max Flow
- Boykov-Kolmogorov Max Flow
- K-core
- Jones-Plassmann Coloring
- Brandes Betweenness Centrality
- (More to come)

Performance: Triangle Counting



- Fully generic library C++ library **matches or exceeds** performance of benchmark code (Beamer GAP benchmark)
- 32-core dual-socket Xeon