Ligra:

A Lightweight Graph Processing Framework for Shared Memory
What's it hoping to achieve?

1. A simple, concise framework

2. High-performance for shared-memory machines
Why?

→ An abundance of graph processing applications

Problems with other, contemporary, graph processing applications:

1. Focus on the distributed case which is often
   a. less efficient per core, per dollar, per watt, etc.
   b. more complex
   c. examples: Boost Graph Library, Pregel, Pegasus, PowerGraph, Knowledge Discovery Toolkit
Relevant Work: Beamer et al’s fast, hybrid BFS implementation for shared memory

1. Combines a:
   a. top-down approach ← *small frontier*
   b. bottom-up approach ← *dense frontiers*
Relevant Work: Beamer et al’s fast, hybrid BFS implementation for shared memory

1. Combines a:
   a. top-down approach $\leftarrow$ small frontier
   b. bottom-up approach $\leftarrow$ dense frontiers
Ligra

A new framework based on Beamer et al’s work

Extends Beamer et al’s idea of a hybrid system to more graphing applications in order to create a lightweight framework for shared memory.
A novel framework

Datatypes:

1. \( G = (V, E) \) (or \( G = (V, E, w(E)) \))
2. \( \text{vertexSubsets} : (U \subseteq V) \)

Functions:

1. \( \text{vertexMap}(U : \text{vertexSubset}, F : \text{vertex} \rightarrow \text{bool}) : \text{vertexSubset} \)
2. \( \text{edgeMap}(G : \text{graph}, U : \text{vertexSubset}, F : (\text{vertex} \times \text{vertex}) \rightarrow \text{bool}, C : \text{vertex} \rightarrow \text{bool}) : \text{vertexSubset} \)
Ligra: Hybridization

**SPARSE:**

→ **vertices:** [0,2,3] or [3,2,0]

→ **edgeMapSparse**
  
  - $F(u, ngh) \ \forall \ ngh \in \text{neighbours}(u)$
  
  - $\propto |U| + \sum \text{outdegrees}(U)$

→ **Switch on** $|U| + \sum \text{outdegrees}(U) > |E|/20$

**DENSE:**

→ **vertices:** [1,0,1,1,0,0,0,0]

→ **edgeMapDense**
  
  - $F(ngh,v) \ \forall \ ngh \in \text{neighbours}(v) \text{ where } v \in U$
  
  - $\propto d|V|$
Ligra: Graph Representation

in-edges: (out-edges similarly)

Vertex: 3
indegree: 3
outdegree: 5
An Example: BFS

Parents = {-1, ..., -1}

procedure Update(s, d)
  return (CAS(&Parents[d], -1, s))

procedure Cond(i)
  return (Parents[i] == -1)

procedure BFS(G, r)
  Parents[r] = r Frontier = {r}
  while (size(Frontier) != 0) do Frontier = edgeMap(G, Frontier, Update, Cond)
An Example: Connected Components

Algorithm 8 Connected Components

1: IDs = \{0, \ldots, |V| - 1\} \quad \triangleright \text{initialized such that IDs}[i] = i
2: prevIDs = \{0, \ldots, |V| - 1\} \quad \triangleright \text{initialized such that prevIDs}[i] = i
3:
4: \textbf{procedure} CCUPDATE(s, d)
5: \quad \text{origID} = \text{IDs}[d]
6: \quad \textbf{if} (\text{WRITEMIN}(&\text{IDs}[d], \text{IDs}[s])) \textbf{then}
7: \quad \quad \text{return} \ (\text{origID} == \text{prevIDs}[d])
8: \quad \text{return} 0
9:
10: \textbf{procedure} COPY(i)
11: \quad \text{prevIDs}[i] = \text{IDs}[i]
12: \quad \text{return} 1
13:
14: \textbf{procedure} CC(G)
15: \quad \text{Frontier} = \{0, \ldots, |V| - 1\} \quad \triangleright \text{vertexSubset initialized to } |\text{V}|
16: \quad \textbf{while} (\text{SIZE}(|\text{Frontier}|) \neq 0) \textbf{do}
17: \quad \quad \text{Frontier} = \text{VERTEXMAP}(|\text{Frontier}|, \text{COPY})
18: \quad \quad \text{Frontier} = \text{EDGEMAP}(G, \text{Frontier}, \text{CCUPDATE}, C_{true})
19: \quad \text{return} \text{IDs}
Evaluation & Experiments

Algorithms:
1. Bellman-Ford
2. PageRank
3. CC, Graph Radii
4. Betweenness Centrality
5. Breadth-First Search

Datasets:
1. 3D-grid
2. random-local
3. rMat24, rMat27
4. Twitter, Yahoo
10-39x speedup from using Ligra on a range of algorithms
1. **Betweeness Centrality**
   a. **KDT**: can traverse $\sim \frac{1}{5}$ the number of edges as Ligra but on a graph that is smaller
   b. **problem**: KDT uses a batch processing system

2. **PageRank**
   a. **GPS**: running time of 1.44 min/iteration whereas **Ligra**: takes 20 sec/iteration on a larger graph
   b. **Powergraph**: running time of 3.6 sec/iterations vs **Ligra**: 2.91 sec/iteration

3. **Connected-Components**
   a. **Pegasus**: running time of 10 min/6 iterations vs **Ligra**: 10 seconds/6 iterations
Problems with Evaluation

1. Comparing similar graphs on similar problems

2. The dramatic improvements are a bit suspect -- XStream paper

3. Is improvement based on clever use of a poorly implemented language (e.g. the authors know lots about the programming language -- but what about the average user)?
Strengths & Weaknesses

Strengths:
- simple idea/easy to use
- can get impressive speedups

Weaknesses:
- Narrow optimisation
- Inconsistent evaluation
- Are the assumptions valid?
Take-away

1. We can use a hybridization method for some optimisations

2. A focus on shared-memory