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

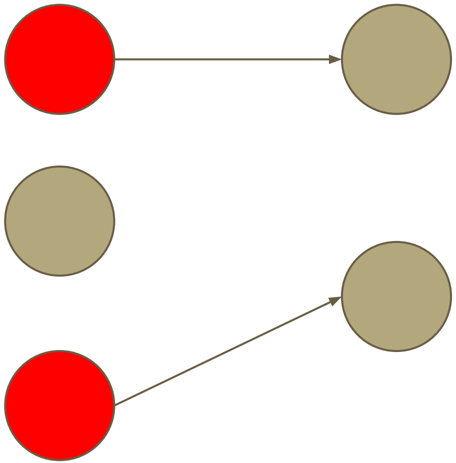
Problems with other, contemporary, graph processing applications:

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

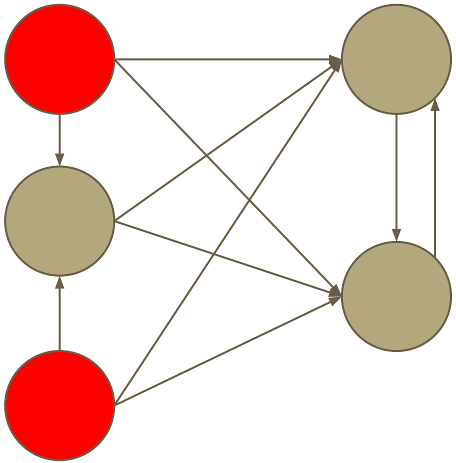
1. Combines a :
    a. top-down approach ← ***small frontier***

    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 ← ***small frontier***
    b. bottom-up approach ← ***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. vertexSubsets : (U $\subseteq$ V)

Functions:

1. **vertexMap**(U : vertexSubset, F : vertex → bool) : vertexSubset
2. **edgeMap**(G : graph, U : vertexSubset, F : (vertex x vertex) → bool, C : vertex → bool) : vertexSubset)

Ligra: Hybridization

SPARSE:

→ **vertices:** [0,2,3] or [3,2,0]

→ **edgeMapSparse**

- F(u,ngh) ∀ ngh ∈ neighbours (u)
- ∝ |U| + ∑ outdegrees(U)

DENSE:

→ **vertices**: [1,0,1,1,0,0,0,0]

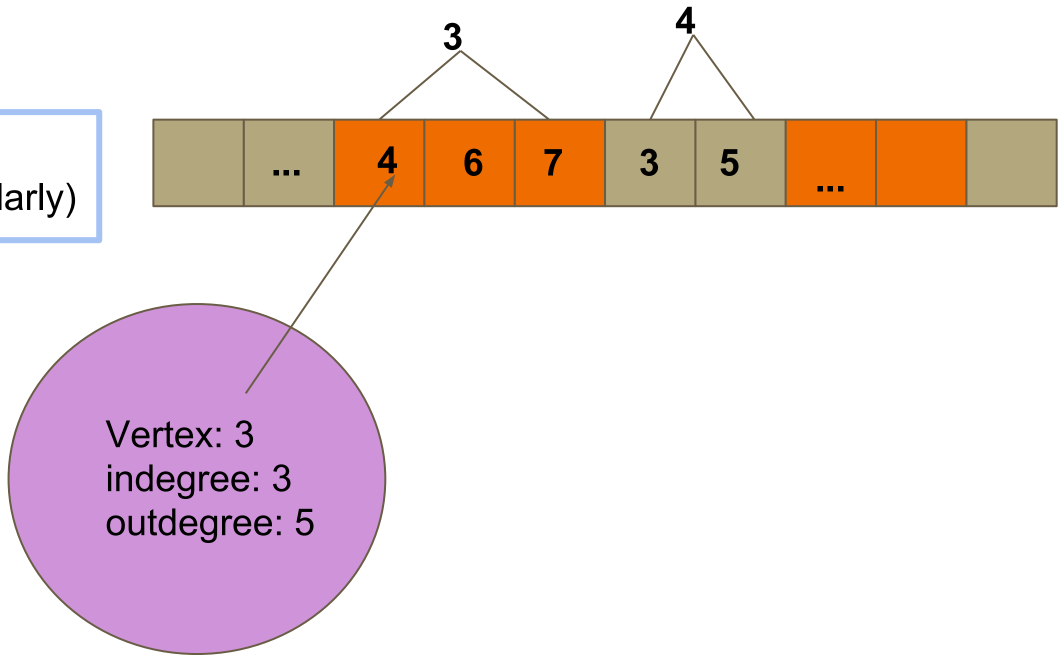
→ **edgeMapDense**

- F(ngh,v) ∀ ngh ∈ neighbours (v) where v ∈ U
- ∝ d|V|

→ Switch on |U| + ∑ outdegrees(U) > |E|/20

Ligra: Graph Representation

in-edges:
(out-edges similarly)



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

```
IDs = {0, ..., |V| − 1}                          ▷ initialized such that IDs[i] = i
prevIDs = {0, ..., |V| − 1}                      ▷ initialized such that prevIDs[i] = i

procedure CCUPDATE(s, d)
    origID = IDs[d]
    if (WRITEMIN(&IDs[d], IDs[s])) then
        return (origID == prevIDs[d])
    return 0

procedure COPY(i)
    prevIDs[i] = IDs[i]
    return 1

procedure CC(G)
    Frontier = {0, ..., |V| − 1}                 ▷ vertexSubset initialized to V
    while (SIZE(Frontier) ≠ 0) do
        Frontier = VERTEXMAP(Frontier, COPY)
        Frontier = EDGEMAP(G, Frontier, CCUPDATE, C_true)
    return 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

Comparative Evaluation

1. **Betweeness Centrality**
    a. **KDT:** can traverse ~⅕ 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 20sec/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 10min/6iterations vs **Ligra**: 10 seconds/6iterations

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