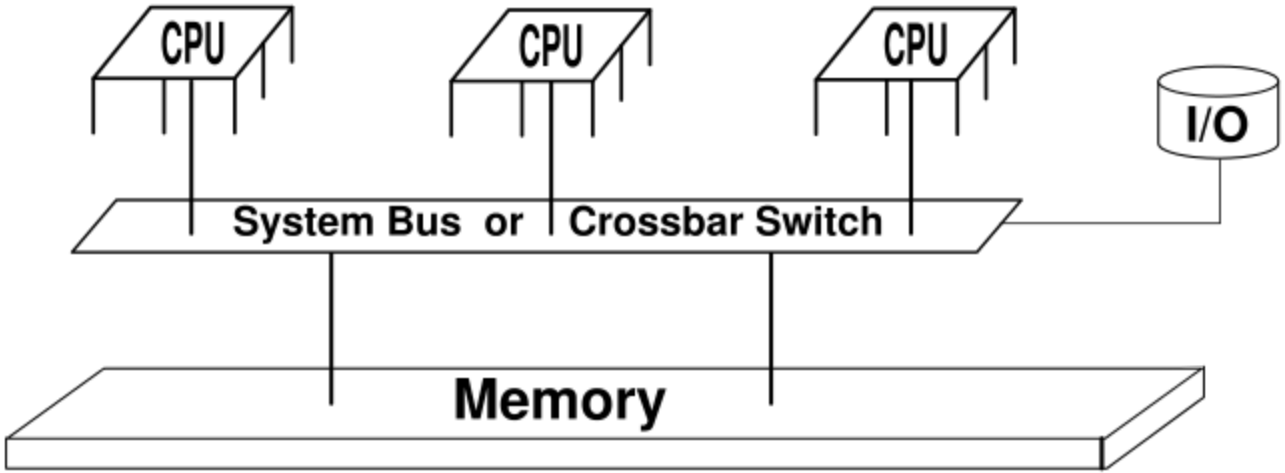
Ligra: A Lightweight Graph Processing Framework for Shared Memory

Shared memory



Other not necessarily SM frameworks

- parallel Boost graph library (PBGL)
- Pregel
- Pegasus
- GraphLab*
- PowerGraph
- Knowledge Discovery Toolkit
- GPS
- Giraph
- Grace*

The Framework

- Written in C++
- Uses CilkPlus (or OpenMP or Intel Math Kernel Library if compiled with icpc)
- Available on GitHub: https://github.com/jshun/ligra

Preliminaries:

- Graph *G(V,E)* or *G(V,E,w)*
- Compare-and-Swap *CAS(&loc, oldV, newV)* return bool
- vertexSubset

Framework operations - EdgeMap

$$\textsc{EdgeMap}(G : \mathit{graph},\ U : \mathit{vertexSubset},\ F : (\mathit{vertex} \times \mathit{vertex}) \mapsto \mathit{bool},\ C : \mathit{vertex} \mapsto \mathit{bool}) : \mathit{vertexSubset}.$$

Algorithm 1 EDGEMAP

```
1: procedure EDGEMAP(G, U, F, C)
2:     if (|U| + sum of out-degrees of U > threshold) then
3:         return EDGEMAPDENSE(G, U, F, C)
4:     else return EDGEMAPSPARSE(G, U, F, C)
```

threshold is |*E*|/20

Algorithm 2 EDGEMAPSPARSE

```
1: procedure EDGEMAPSPARSE(G, U, F, C)
2:     Out = {}
3:     parfor each v ∈ U do
4:         parfor ngh ∈ N+(v) do
5:             if (C(ngh) == 1 and F(v, ngh) == 1) then
6:                 Add ngh to Out
7:     Remove duplicates from Out
8:     return Out
```

Algorithm 3 EDGEMAPDENSE

```
1: procedure EDGEMAPDENSE(G, U, F, C)
2:     Out = {}
3:     parfor i ∈ {0, ..., |V| − 1} do
4:         if (C(i) == 1) then
5:             for ngh ∈ N−(i) do
6:                 if (ngh ∈ U and F(ngh, i) == 1) then
7:                     Add i to Out
8:                 if (C(i) == 0) then break
9:     return Out
```

Algorithm 5 EDGEMAPDENSE-WRITE

```
1: procedure EDGEMAPDENSE-WRITE(G, U, F, C)
2:     Out = {}
3:     parfor i ∈ {0, ..., |V| − 1} do
4:         if (i ∈ U) then
5:             parfor ngh ∈ N+(i) do
6:                 if (C(ngh) == 1 and F(i, ngh) == 1) then
7:                     Add ngh to Out
8:     return Out
```

Framework operations - VertexMap

$$\textbf{VERTEXMAP}(U : vertexSubset, F : vertex \mapsto bool) : vertexSubset.$$

Algorithm 4 VERTEXMAP

```
1: procedure VERTEXMAP(U, F)
2:     Out = {}
3:     parfor u ∈ U do
4:         if (F(u) == 1) then Add u to Out
5:     return Out
```

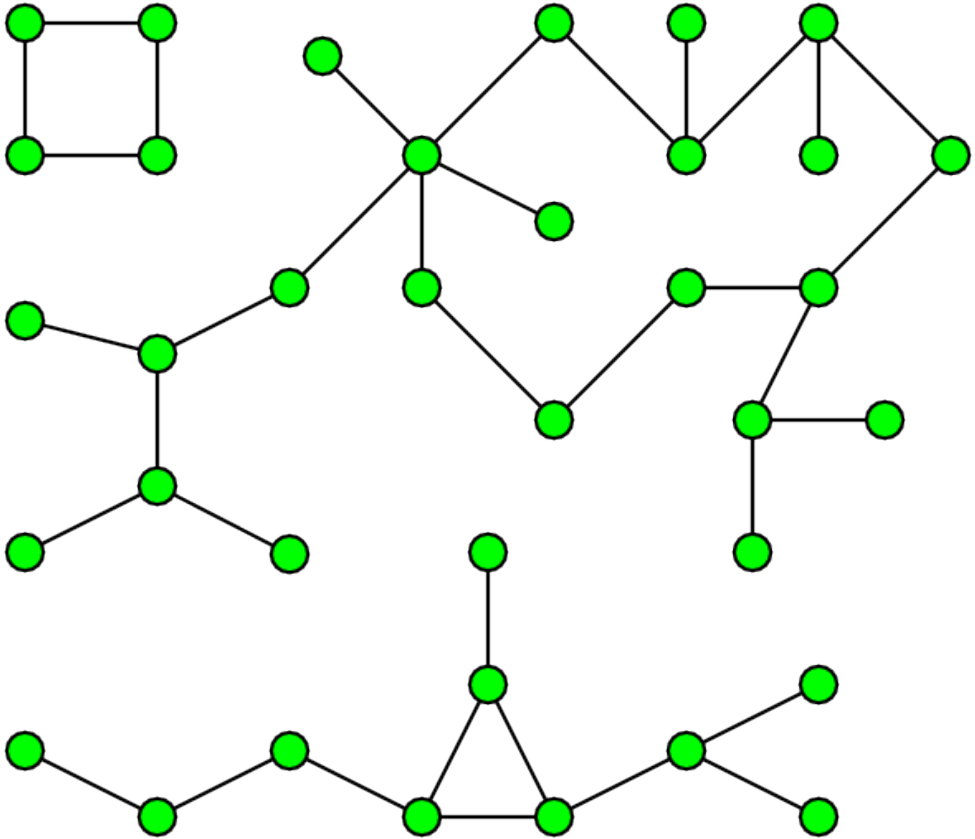
Framework - Applications

- **Breadth-First Search**
- Betweenness Centrality
- Graph Radii
- **Connected Components**
- PageRank
- **Bellman-Ford**

BFS

```
1:  Parents = {−1, . . . , −1}                          ▷ initialized to all -1's
2:
3:  procedure UPDATE(s, d)
4:      return (CAS(&Parents[d], −1 , s ))
5:
6:  procedure COND(i)
7:      return (Parents[i] == −1)
8:
9:  procedure BFS(G, r)                                 ▷ r is the root
10:     Parents[r] = r
11:     Frontier = {r}          ▷ vertexSubset initialized to contain only r
12:     while (SIZE(Frontier) ≠ 0) do
13:         Frontier = EDGEMAP(G, Frontier, UPDATE, COND)
```

Connected Components



Algorithm 8 Connected Components

```
1: IDs = {0, ..., |V| − 1}                          ▷ initialized such that IDs[i] = i
2: prevIDs = {0, ..., |V| − 1}                      ▷ initialized such that prevIDs[i] = i
3:
4: procedure CCUPDATE(s, d)
5:     origID = IDs[d]
6:     if (WRITEMIN(&IDs[d], IDs[s])) then
7:         return (origID == prevIDs[d])
8:     return 0
9:
10: procedure COPY(i)
11:     prevIDs[i] = IDs[i]
12:     return 1
13:
14: procedure CC(G)
15:     Frontier = {0, ..., |V| − 1}                ▷ vertexSubset initialized to V
16:     while (SIZE(Frontier) ≠ 0) do
17:         Frontier = VERTEXMAP(Frontier, COPY)
18:         Frontier = EDGEMAP(G, Frontier, CCUPDATE, C_true)
19:     return IDs
```

Algorithm 10 Bellman-Ford

```
1:  SP = {∞, ..., ∞}                                   ▷ initialized to all ∞
2:  Visited = {0, ..., 0}                              ▷ initialized to all 0
3:
4:  procedure BFUPDATE(s, d, edgeWeight)
5:      if (WRITEMIN(&SP[d], SP[s] + edgeWeight)) then
6:          return CAS(&Visited[d], 0, 1)
7:      else return 0
8:
9:  procedure BFRESET(i)
10:     Visited[i] = 0
11:     return 1
12:
13: procedure BELLMAN-FORD(G, r)
14:     SP[r] = 0
15:     Frontier = {r}          ▷ vertexSubset initialized to contain just r
16:     round = 0
17:     while (SIZE(Frontier) ≠ 0 and round < |V|) do
18:         round = round + 1
19:         Frontier = EDGEMAP(G, Frontier, BF-UPDATE, C_true)
20:         Frontier = VERTEXMAP(Frontier, BF-RESET)
21:     if (round == |V|) then return "negative-weight cycle"
22:     else return SP
```

Performance - the graphs

Input	Num. Vertices	Num. Directed Edges
3D-grid	10^7	6×10^7
random-local	10^7	9.8×10^7
rMat24	1.68×10^7	9.9×10^7
rMat27	1.34×10^8	2.12×10^9
Twitter	4.17×10^7	1.47×10^9
Yahoo*	1.4×10^9	12.9×10^9

Table 1. Graph inputs. *The original asymmetric graph has 6.6×10^9 edges.

Performance - running times

Application	3D-grid			random-local			rMat24			rMat27			Twitter			Yahoo		
	(1)	(40h)	(SU)	(1)	(40h)	(SU)	(1)	(40h)	(SU)	(1)	(40h)	(SU)	(1)	(40h)	(SU)	(1)	(40h)	(SU)
Breadth-First Search	2.9	0.28	10.4	2.11	0.073	28.9	2.83	0.104	27.2	11.8	0.423	27.9	6.92	0.321	21.6	173	8.58	20.2
Betweenness Centrality	9.15	0.765	12.0	8.53	0.265	32.2	11.3	0.37	30.5	113	4.07	27.8	47.8	2.64	18.1	634	23.1	27.4
Graph Radii	351	10.0	35.1	25.6	0.734	34.9	39.7	1.21	32.8	337	12.0	28.1	171	7.39	23.1	1280	39.6	32.3
Connected Components	51.5	1.71	30.1	14.8	0.399	37.1	14.1	0.527	26.8	204	10.2	20.0	78.7	3.86	20.4	609	29.7	20.5
PageRank (1 iteration)	4.29	0.145	29.6	6.55	0.224	29.2	8.93	0.25	35.7	243	6.13	39.6	72.9	2.91	25.1	465	15.2	30.6
Bellman-Ford	63.4	2.39	26.5	18.8	0.677	27.8	17.8	0.694	25.6	116	4.03	28.8	75.1	2.66	28.2	255	14.2	18.0

Table 2. Running times (in seconds) of algorithms over various inputs on a 40-core machine (with hyper-threading). (SU) indicates the speedup of the application (single-thread time divided by 40-core time).