Ligra: A Lightweight Graph Processing Framework for Shared Memory

Julian Shun and Guy Blelloch 2013

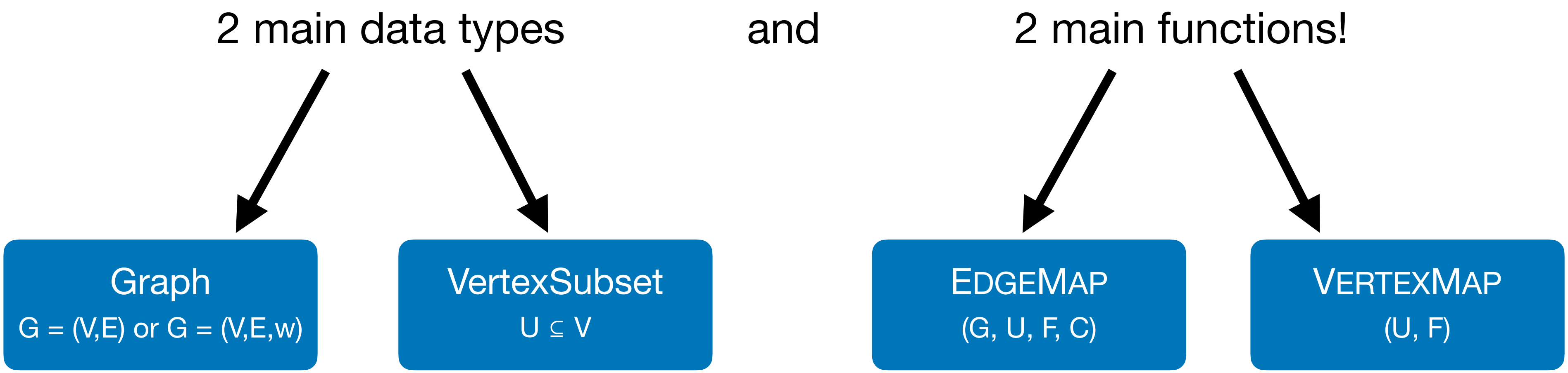
Presented by Sarah Zhao, R244 10.26.2022

Motivation

1. Shared Memory vs Distributed Memory
    - Many graph-processing frameworks designed for distributed memory systems (e.g. Pregel)
    - Advancements in technology -> enough storage in shared-memory machines (can handle graphs 100 billion edges in main memory)
    - Data locality, cheaper communication costs
2. Beamer et al, 2011 and 2012: hybrid approach to BFS exploiting variation in number of vertices and edges (i.e. frontier size) computed in each iteration of a parallel process
    - Can we generalize to other algorithms?

What is Ligra?

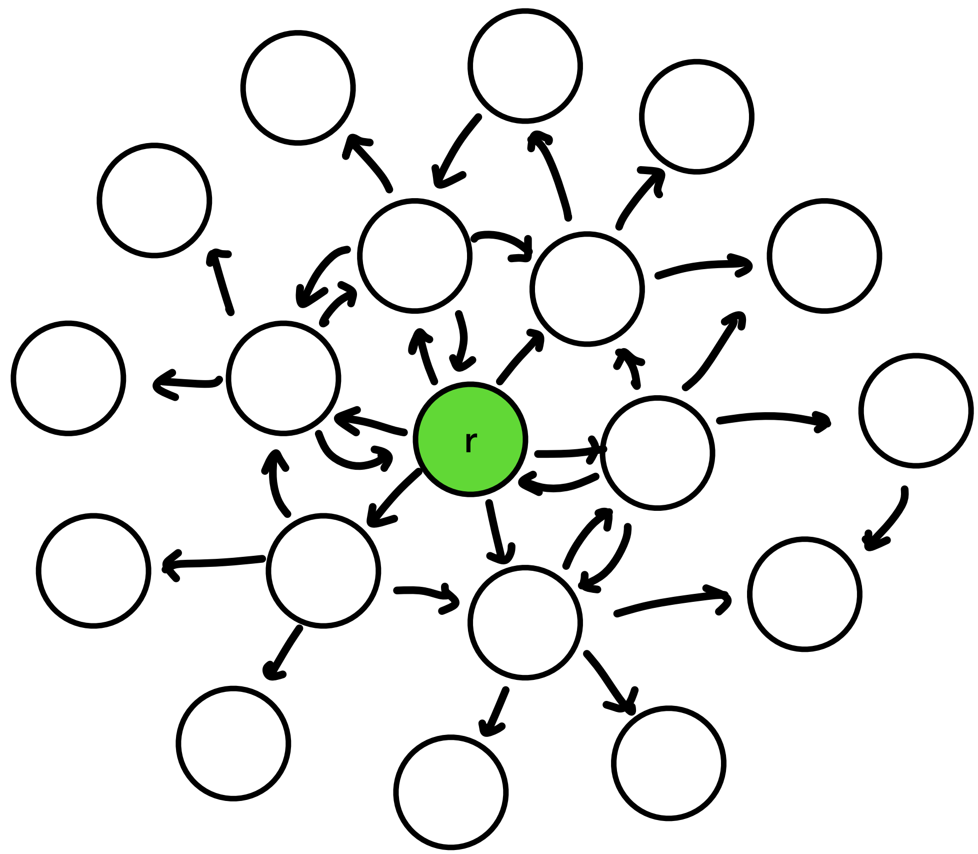
Simple shared-memory parallel graph-processing framework:



Application: BFS

```
Parents = {−1, . . . , −1}                          ▷ initialized to all -1's

procedure UPDATE(s, d)
    return (CAS(&Parents[d], −1 , s ))

procedure COND(i)
    return (Parents[i] == −1)

procedure BFS(G, r)                                  ▷ r is the root
    Parents[r] = r
    Frontier = {r}          ▷ vertexSubset initialized to contain only r
    while (SIZE(Frontier) ≠ 0) do
        Frontier = EDGEMAP(G, Frontier, UPDATE, COND)
```

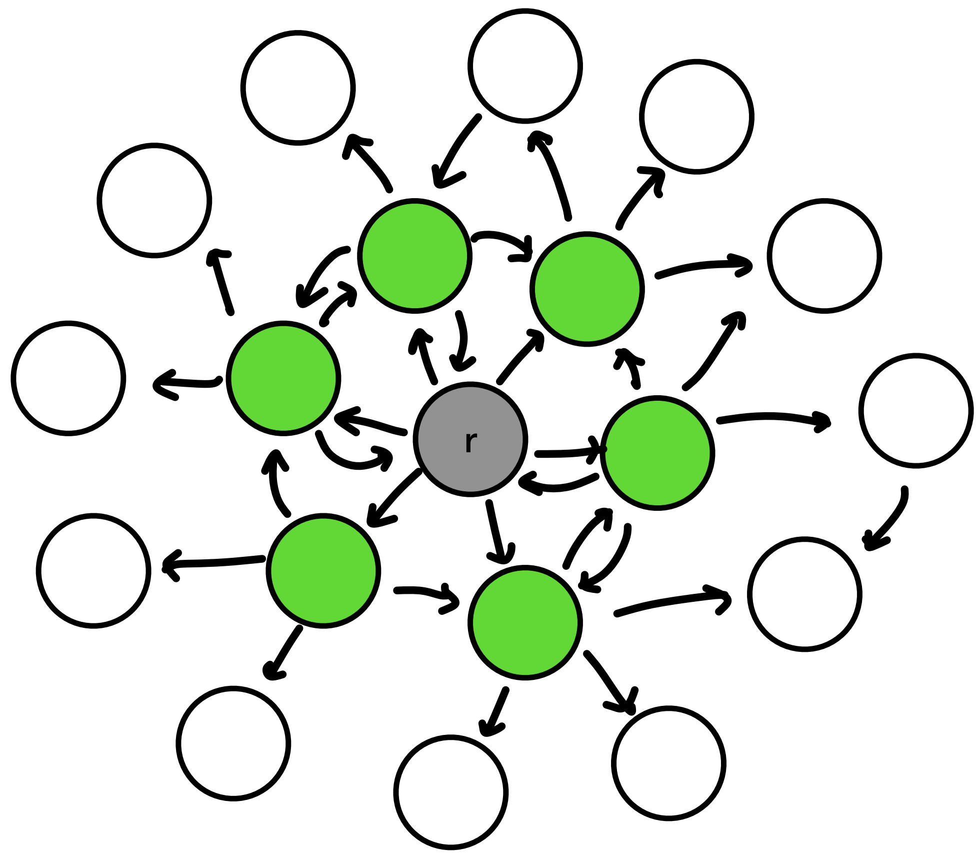
Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(*loc*,*oldV*,*newV*) atomically checks if the value at location *loc* is equal to *oldV* and if so it updates *loc* with *newV* and returns *true*. Otherwise it leaves *loc* unmodified and returns *false*.



Application: BFS

```
Parents = {−1, . . . , −1}                         ▷ initialized to all -1's

procedure UPDATE(s, d)
    return (CAS(&Parents[d], −1 , s ))

procedure COND(i)
    return (Parents[i] == −1)

procedure BFS(G, r)                                ▷ r is the root
    Parents[r] = r
    Frontier = {r}              ▷ vertexSubset initialized to contain only r
    while (SIZE(Frontier) ≠ 0) do
        Frontier = EDGEMAP(G, Frontier, UPDATE, COND)
```

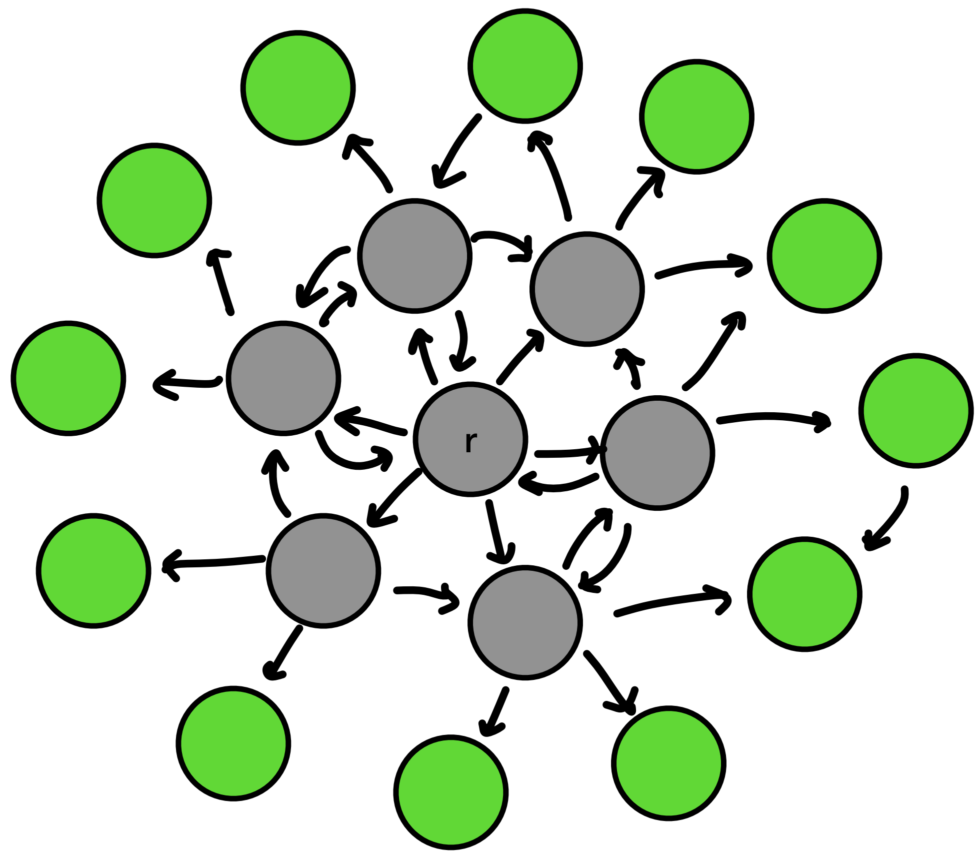
Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(*loc*,*oldV*,*newV*) atomically checks if the value at location *loc* is equal to *oldV* and if so it updates *loc* with *newV* and returns *true*. Otherwise it leaves *loc* unmodified and returns *false*.



Application: BFS

```
Parents = {−1, . . . , −1}                                   ▷ initialized to all -1's

procedure UPDATE(s, d)
    return (CAS(&Parents[d], −1 , s ))

procedure COND(i)
    return (Parents[i] == −1)

procedure BFS(G, r)                                                  ▷ r is the root
    Parents[r] = r
    Frontier = {r}            ▷ vertexSubset initialized to contain only r
    while (SIZE(Frontier) ≠ 0) do
        Frontier = EDGEMAP(G, Frontier, UPDATE, COND)
```

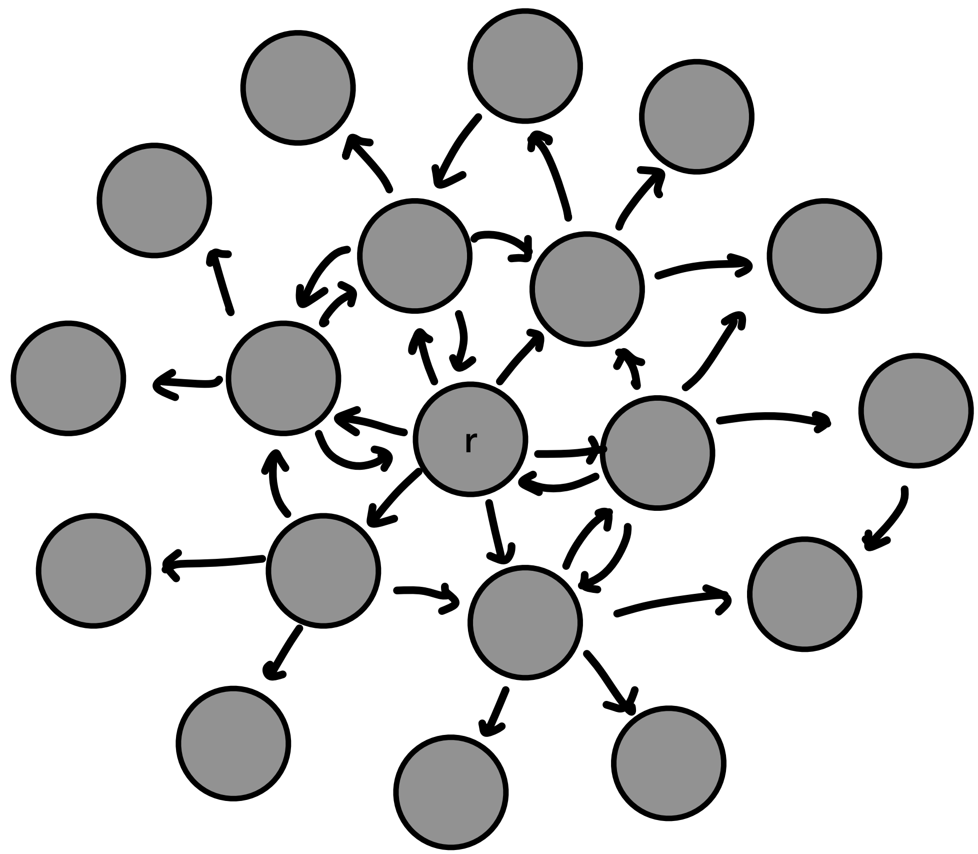
Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(*loc*,*oldV*,*newV*) atomically checks if the value at location *loc* is equal to *oldV* and if so it updates *loc* with *newV* and returns *true*. Otherwise it leaves *loc* unmodified and returns *false*.



Application: BFS

```
Parents = {−1, . . . , −1}                          ▷ initialized to all -1's

procedure UPDATE(s, d)
    return (CAS(&Parents[d], −1 , s ))

procedure COND(i)
    return (Parents[i] == −1)

procedure BFS(G, r)                                  ▷ r is the root
    Parents[r] = r
    Frontier = {r}          ▷ vertexSubset initialized to contain only r
    while (SIZE(Frontier) ≠ 0) do
        Frontier = EDGEMAP(G, Frontier, UPDATE, COND)
```

Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(*loc*,*oldV*,*newV*) atomically checks if the value at location *loc* is equal to *oldV* and if so it updates *loc* with *newV* and returns *true*. Otherwise it leaves *loc* unmodified and returns *false*.



Hybrid Model for Varying Frontier-Size

Algorithm 1 EDGEMAP

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

Algorithm 4 VERTEXMAP

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

Algorithm 2 EDGEMAPSPARSE

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

Check out-neighbors for each vertex in the frontier

Algorithm 3 EDGEMAPDENSE

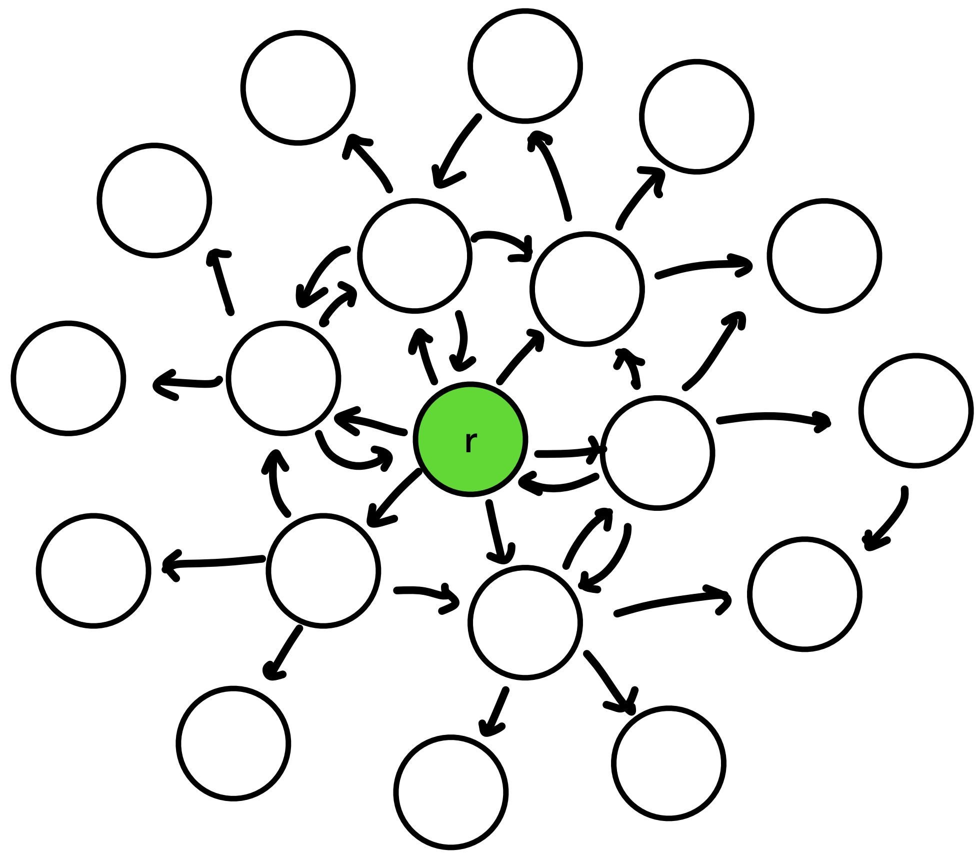
```
procedure EDGEMAPDENSE(G, U, F, C)
    Out = {}
    parfor i ∈ {0, . . . , |V| − 1} do
        if (C(i) == 1) then
            for ngh ∈ N−(i) do
                if (ngh ∈ U and F(ngh, i) == 1) then
                    Add i to Out
                if (C(i) == 0) then break
    return Out
```

Check in-neighbors for each target vertex

Application: BFS

```
Parents = {−1, . . . , −1}                                   ▷ initialized to all -1's

procedure UPDATE(s, d)
    return (CAS(&Parents[d], −1 , s ))

procedure COND(i)
    return (Parents[i] == −1)

procedure BFS(G, r)                                          ▷ r is the root
    Parents[r] = r
    Frontier = {r}              ▷ vertexSubset initialized to contain only r
    while (SIZE(Frontier) ≠ 0) do
        Frontier = EDGEMAP(G, Frontier, UPDATE, COND)
```

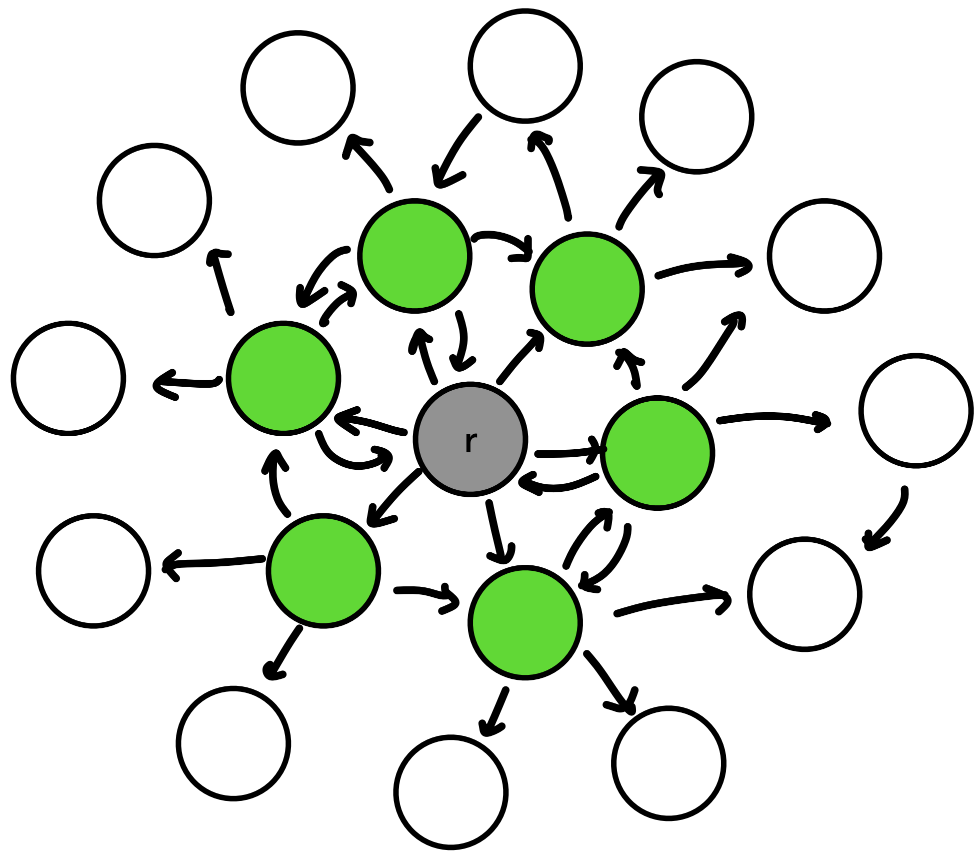
Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(*loc*,*oldV*,*newV*) atomically checks if the value at location *loc* is equal to *oldV* and if so it updates *loc* with *newV* and returns *true*. Otherwise it leaves *loc* unmodified and returns *false*.



Application: BFS

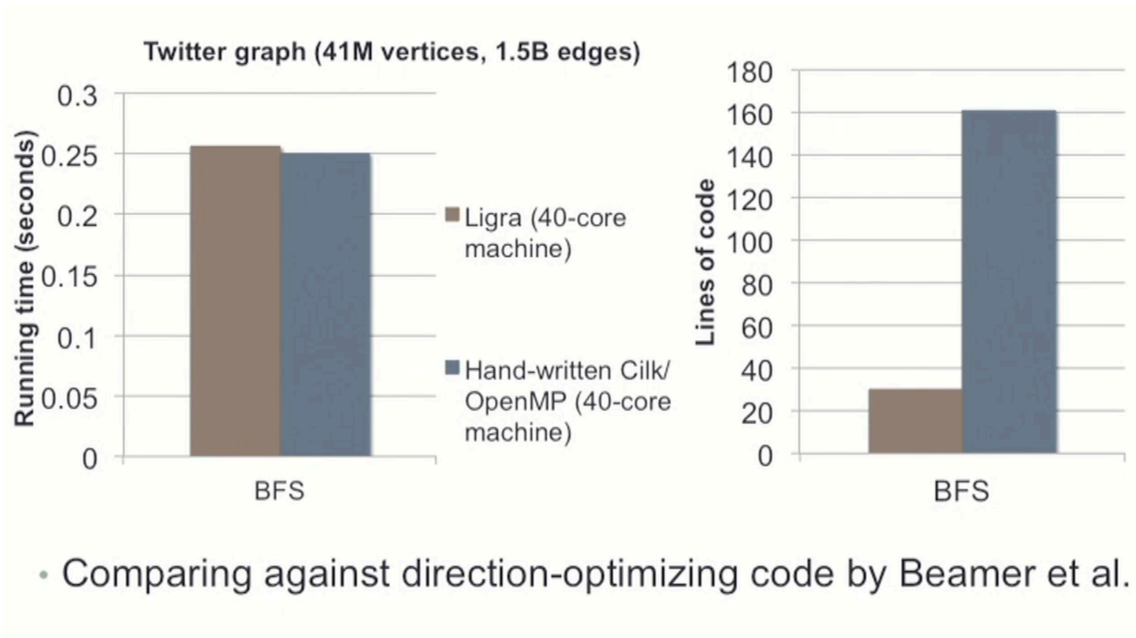
```
Parents = {−1, . . . , −1}                              ▷ initialized to all -1's

procedure UPDATE(s, d)
    return (CAS(&Parents[d], −1 , s ))

procedure COND(i)
    return (Parents[i] == −1)

procedure BFS(G, r)                                     ▷ r is the root
    Parents[r] = r
    Frontier = {r}              ▷ vertexSubset initialized to contain only r
    while (SIZE(Frontier) ≠ 0) do
        Frontier = EDGEMAP(G, Frontier, UPDATE, COND)
```

Figure 1. Pseudocode for Breadth-First Search in our framework. The compare-and-swap function CAS(*loc*,*oldV*,*newV*) atomically checks if the value at location *loc* is equal to *oldV* and if so it updates *loc* with *newV* and returns *true*. Otherwise it leaves *loc* unmodified and returns *false*.



BFS Evaluation

- Implemented on a 40-core Intel machine with 256GB of RAM (and with multithreading)



https://www.youtube.com/watch?v=W5mDx_G45RQ&ab_channel=MMDSFoundation

Other Applications and Results

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

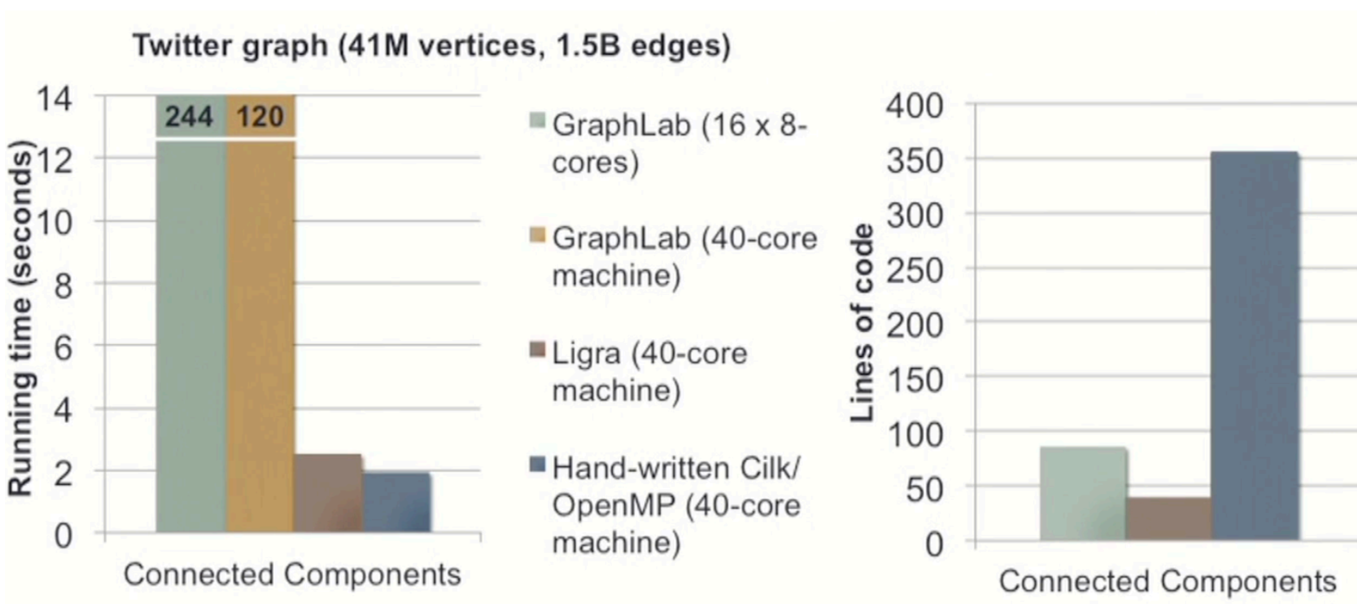
Table 1. Graph inputs. *The original asymmetric graph has 6.6×10^9 edges.

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).

Experimental Results

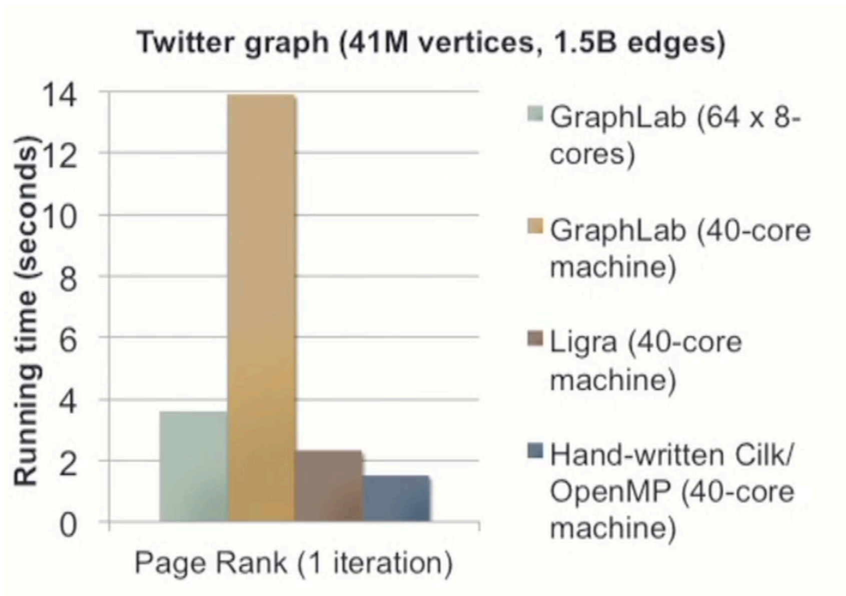
Connected Components

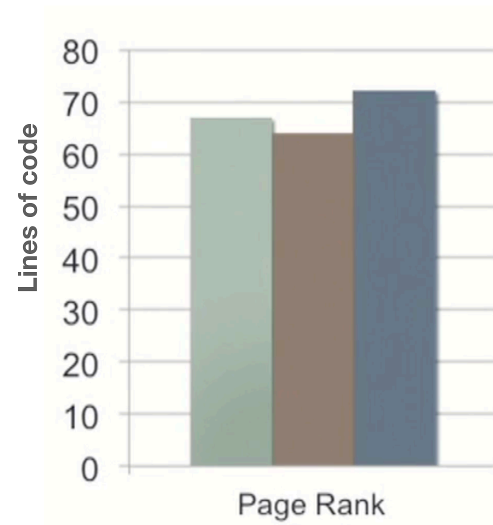


https://www.youtube.com/watch?v=W5mDx_G45RQ&ab_channel=MMDSFoundation

Experimental Results

PageRank





https://www.youtube.com/watch?v=W5mDx_G45RQ&ab_channel=MMDSFoundation

Summary

- Lightweight parallel graph processing framework
- Dependence on shared-memory systems (no communication overhead as compared to distributed systems)
- Designed for frontier-based algorithms
- Comparable to or outperformed then graph-processing frameworks

Discussion

- Incomplete evaluation
- Performance to be hardware dependent
    - Worse performance with a different set-up (64-core AMD Opteron machine)
- Scalability and speedup limited by tech
    - Expansion to GPUs? [Shun et al. 2013]
- Exploring applications in other graph algorithms (e.g. max flow, biconnected components, belief propagation, Markov clustering) [Shun et al. 2013]
- Dynamic graph data, graph data stream processing?

Extensions

- Ligra+ (2015) - compressed graphs -> require less memory

- Julienne (2017) - bucketing-based algorithms (generalization of frontier-based algorithms)

- Hygra (2020) - support for hyper graphs

Code at https://github.com/jshun/ligra

Thank you!

Questions?

References

Shun, Julian, and Guy E. Blelloch. "Ligra: a lightweight graph processing framework for shared memory." In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 135-146. 2013.

Shun, Julian. “Framework for Processing Large Graphs in Shared Memory.” https://www.youtube.com/watch?v=W5mDx_G45RQ&ab_channel=MMDSFoundation. 2016.

Beamer, Scott, Krste Asanovic, David Patterson, Scott Beamer, and David Patterson. "Searching for a parent instead of fighting over children: A fast breadth-first search implementation for graph500." *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2011-117* (2011).

Beamer, Scott, Krste Asanovic, and David Patterson. "Direction-optimizing breadth-first search." In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1-10. IEEE, 2012.

Shun, Julian, Laxman Dhulipala, and Guy E. Blelloch. "Smaller and faster: Parallel processing of compressed graphs with Ligra+." In *2015 Data Compression Conference*, pp. 403-412. IEEE, 2015.

Dhulipala, Laxman, Guy Blelloch, and Julian Shun. "Julienne: A framework for parallel graph algorithms using work-efficient bucketing." In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 293-304. 2017.

Shun, Julian. "Practical parallel hypergraph algorithms." In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 232-249. 2020.