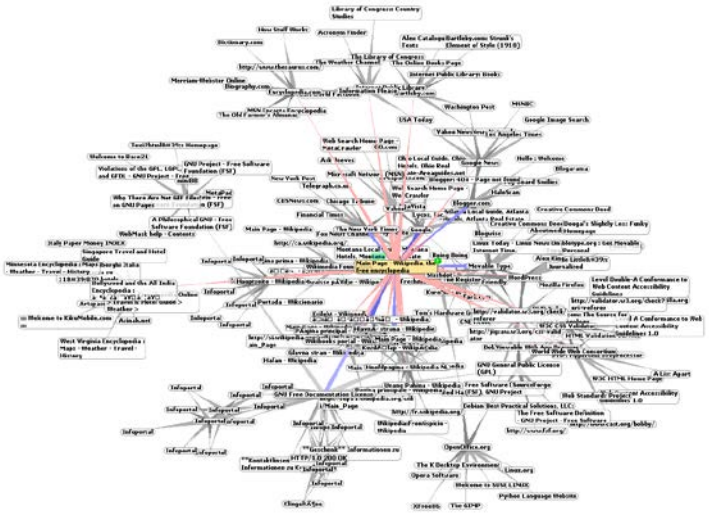
Lightweight Processing on Compressed Graphs

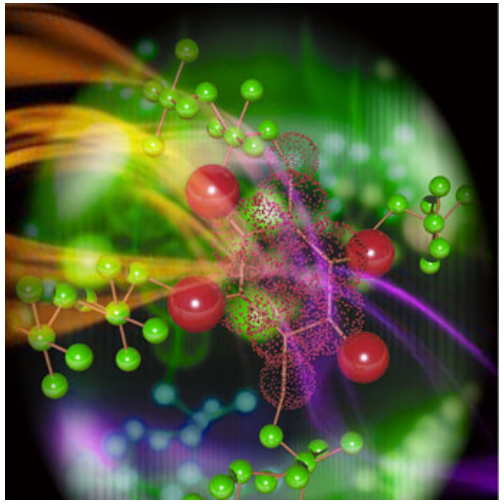
Guy Blelloch
Carnegie Mellon University

Joint work with Julian Shun and Laxman Dhulipala

http://www.istc-cc.cmu.edu/

Large Graphs



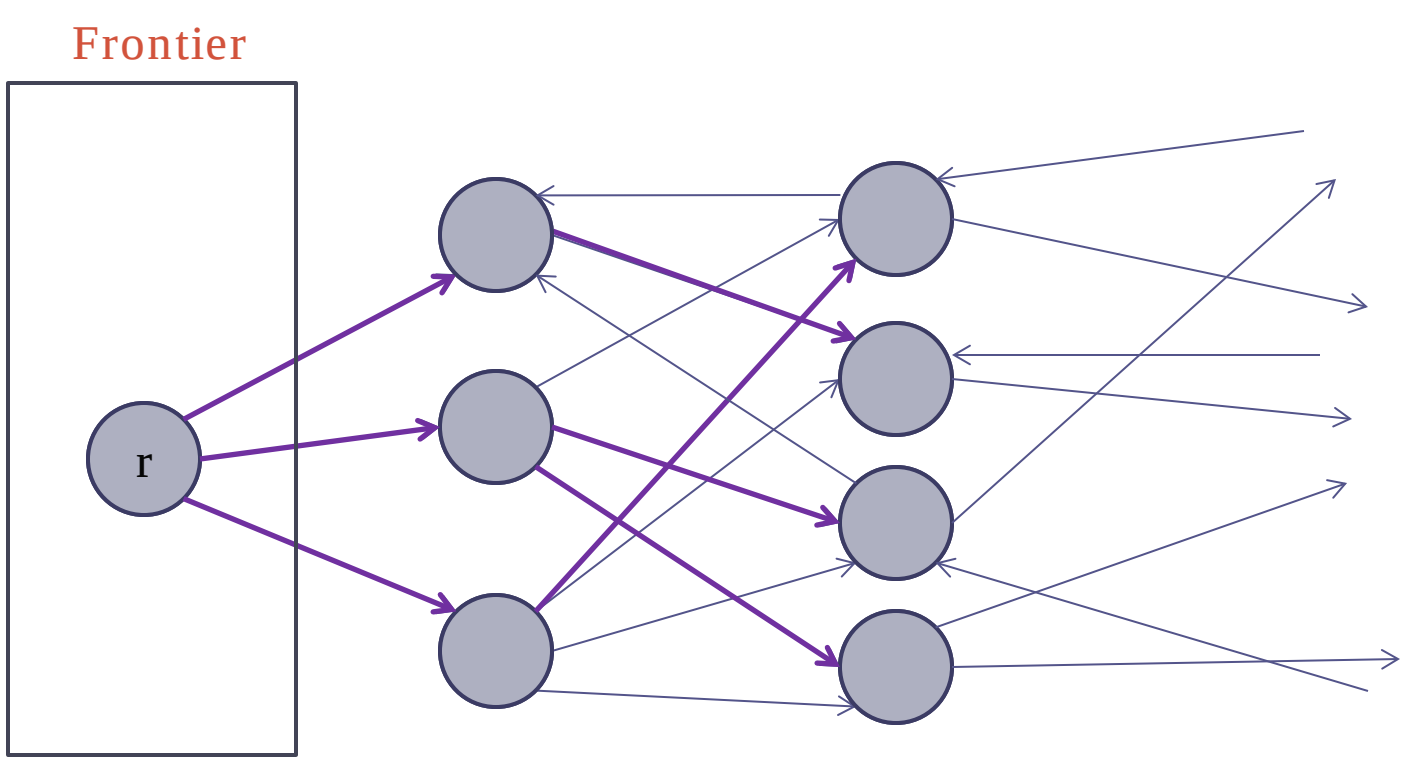


Need to efficiently analyze these graphs

- Computational efficiency
- Space efficiency
- Programming efficiency

Breadth-first Search (BFS)

- Compute a BFS tree rooted at source *r* containing all vertices reachable from *r*



- Can process each frontier in parallel
  - Race conditions, load balancing

BFS Abstractly: Frontier Based

1. Operate on a subset of vertices
2. Map computation over subset of edges in parallel
3. Return new subset of vertices
4. (Map computation over subset of vertices in parallel)

BFS visits every vertext once, but in general can visit many times. Synchronous.

Breadth-first search	Bellman-Ford shortest paths
Betweenness centrality	Graph eccentricity estimation
Connected components	PageRank
Delta stepping	Diameter estimation

Can we build an abstraction for these types of algorithms?

Graph Processing Systems

- Existing: Pregel/Giraph, GraphLab, Pegasus, Knowledge Discovery Toolbox, GraphChi, Parallel BGL, and many others...

- Our system: Ligra - Lightweight graph processing system for shared memory
  - Efficient for "frontier-based" algorithms

- Probably no one-size fits all

Why the Cloud

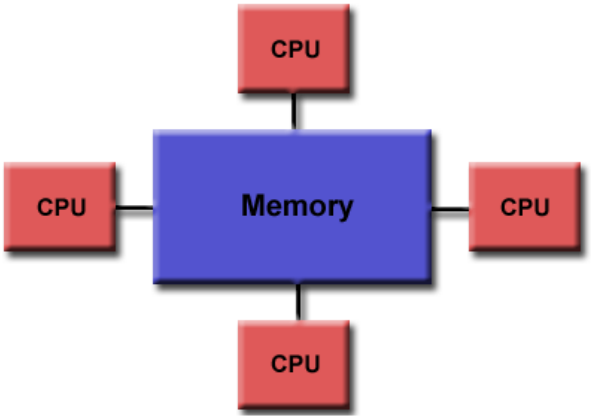
- Costs
- Clould will enable wide use

AWS	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
Memory Optimized - Current Generation					
r3.large	2	6.5	15	1 x 32 SSD	$0.175 per Hour
r3.xlarge	4	13	30.5	1 x 80 SSD	$0.350 per Hour
r3.2xlarge	8	26	61	1 x 160 SSD	$0.700 per Hour
r3.4xlarge	16	52	122	1 x 320 SSD	$1.400 per Hour
r3.8xlarge	32	104	244	2 x 320 SSD	$2.800 per Hour

Ligra

Graph

- Operate on a subset of vertices ← VertexSubset
- Map computation over subset of edges in parallel and return new subset of vertices } EdgeMap
- (Map computation over subset of vertices in parallel) VertexMap



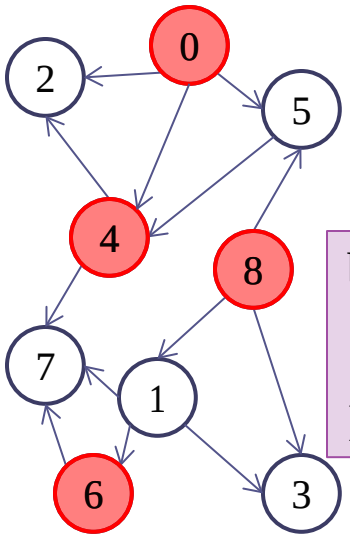
Shared memory

Intel ®
Cilk™ Plus

OpenMP

Cilk Plus/OpenMP

Ligra Framework



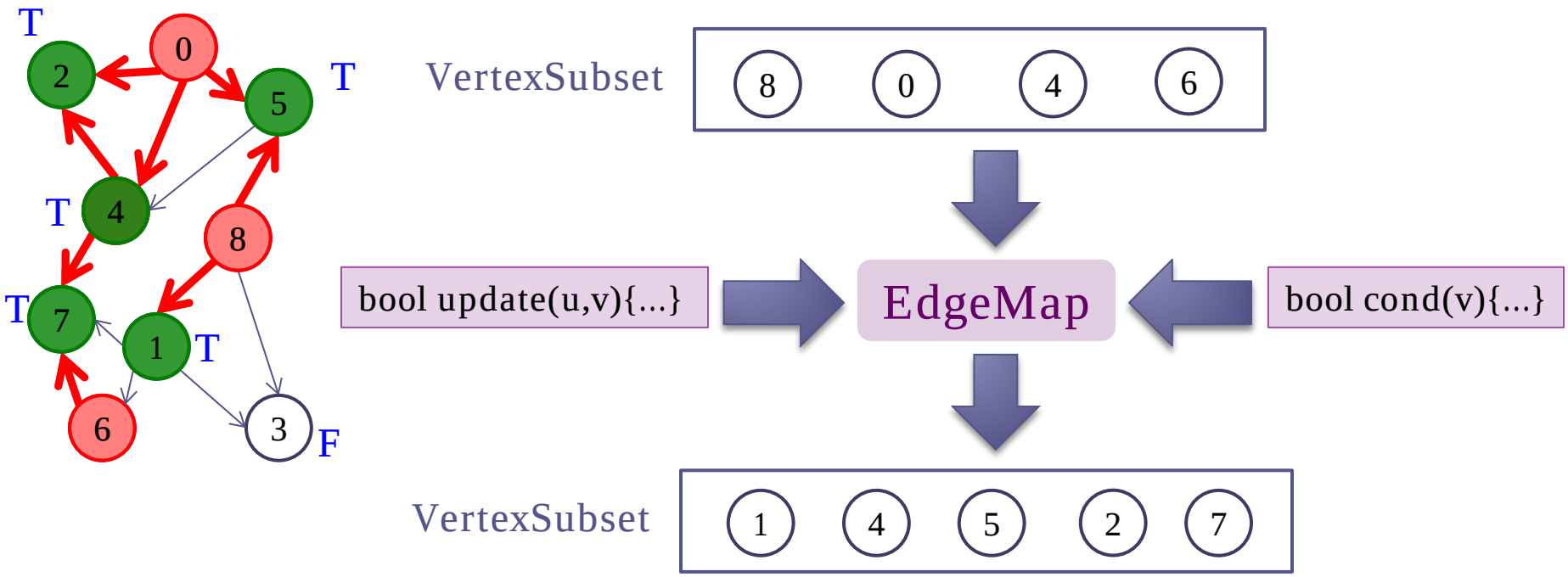

```
bool f(v){
    data[v] = data[v] + 1;
    return (data[v] == 1);
}
```

VertexSubset: 8 0 4 6

VertexMap

VertexSubset: 8 6

Ligra Framework



Why edge based?

- Parallel over the edges
- Sparse/dense (discussed later)

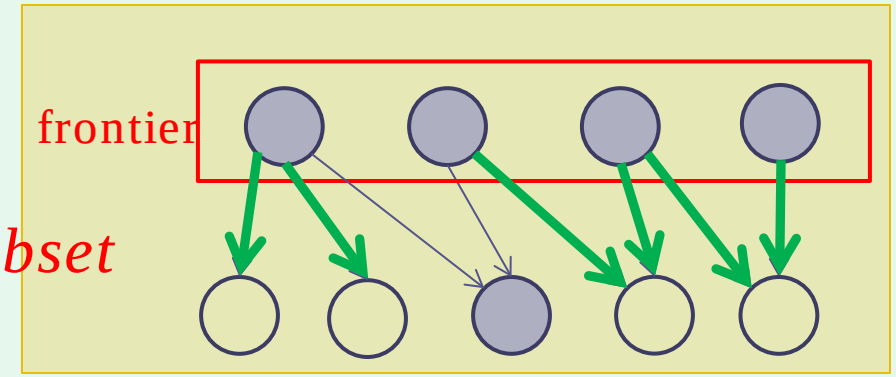
Breadth-first Search in Ligra

```
parents = {-1, ..., -1};   //-1 indicates "unvisited"

procedure UPDATE(s, d):
    return compare_and_swap(parents[d], -1, s);

procedure COND(i):
    return parents[i] == -1;   //checks if "unvisited"

procedure BFS(G, r):
    parents[r] = r;
    frontier = {r}; //VertexSubset
    while (size(frontier) > 0):
            frontier = EDGEMAP(G, frontier, UPDATE,
COND);
```



Actual BFS code in Ligra

```
#include "ligra.h"

struct BFS_F {
  intT* Parents;
  BFS_F(intT* _Parents) : Parents(_Parents) {}
  inline bool update (intT s, intT d) { //Update
    if(Parents[d] == -1) { Parents[d] = s; return 1; }
    else return 0;
  }
  inline bool updateAtomic (intT s, intT d){ //atomic version of Update
    return (CAS(&Parents[d],(intT)-1,s));
  }
  //cond function checks if vertex has been visited yet
  inline bool cond (intT d) { return (Parents[d] == -1); }
};

template <class vertex>
void Compute(graph<vertex> GA, intT start) {
  intT n = GA.n;
  //creates Parents array, initialized to all -1, except for start
  intT* Parents = newA(intT,GA.n);
  parallel_for(intT i=0;i<GA.n;i++) Parents[i] = -1;
  Parents[start] = start;

  vertexSubset Frontier(n,start); //creates initial frontier

  while(!Frontier.isEmpty()){ //loop until frontier is empty
    vertexSubset output = edgeMap(GA, Frontier, BFS_F(Parents));
    Frontier.del();
    Frontier = output; //set new frontier
  }
  Frontier.del();
  free(Parents);
}
```

EdgeMap: Sparse and Dense

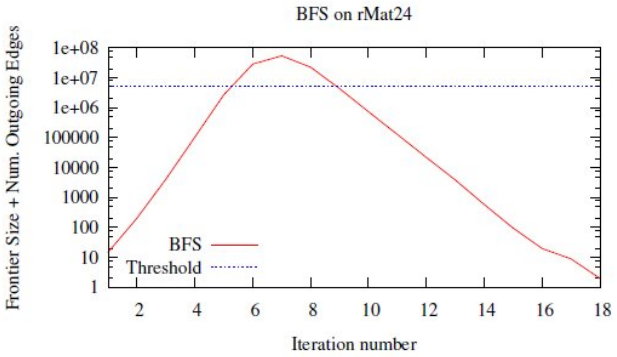
```
procedure EDGEMAP(G, frontier, Update, Cond):
    if (|frontier| + sum of out-degrees > threshold) then:
        return EDGEMAP_DENSE(G, frontier, Update, Cond);
    else:
        return EDGEMAP_SPARSE(G, frontier, Update, Cond);
```

Loop through outgoing edges of frontier vertices in parallel

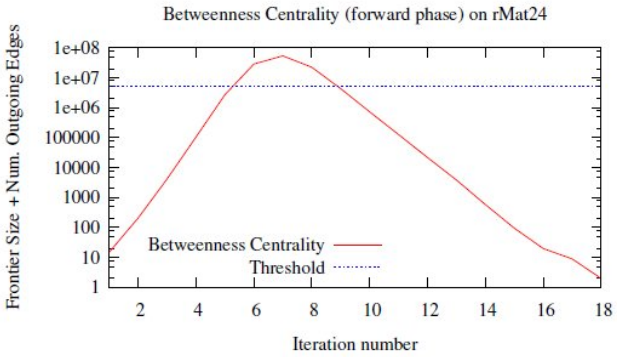
Loop through incoming edges of "unexplored" vertices (in parallel), breaking early if possible

- First used by Beemer for BFS, but Ligra shows that useful for a wide variety of algorithms

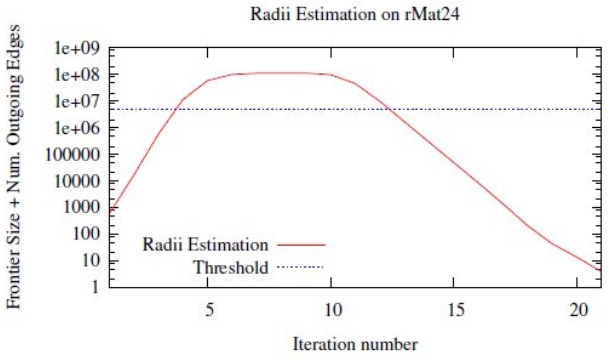
Frontier Plots



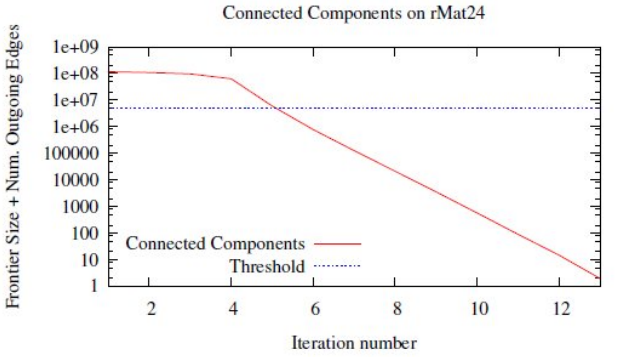
(a) BFS



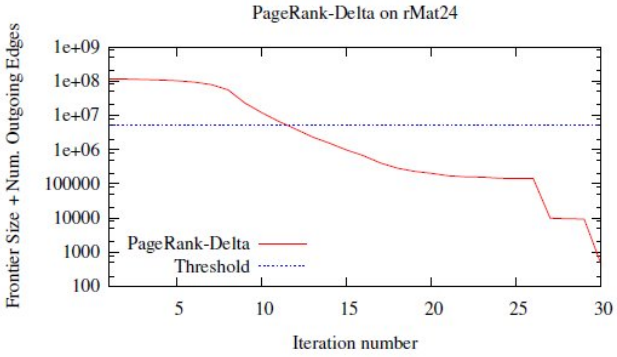
(b) Betweenness Centrality



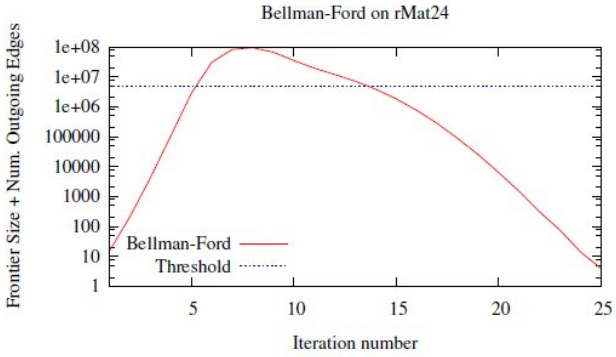
(c) Radii Estimation



(d) Connected Components

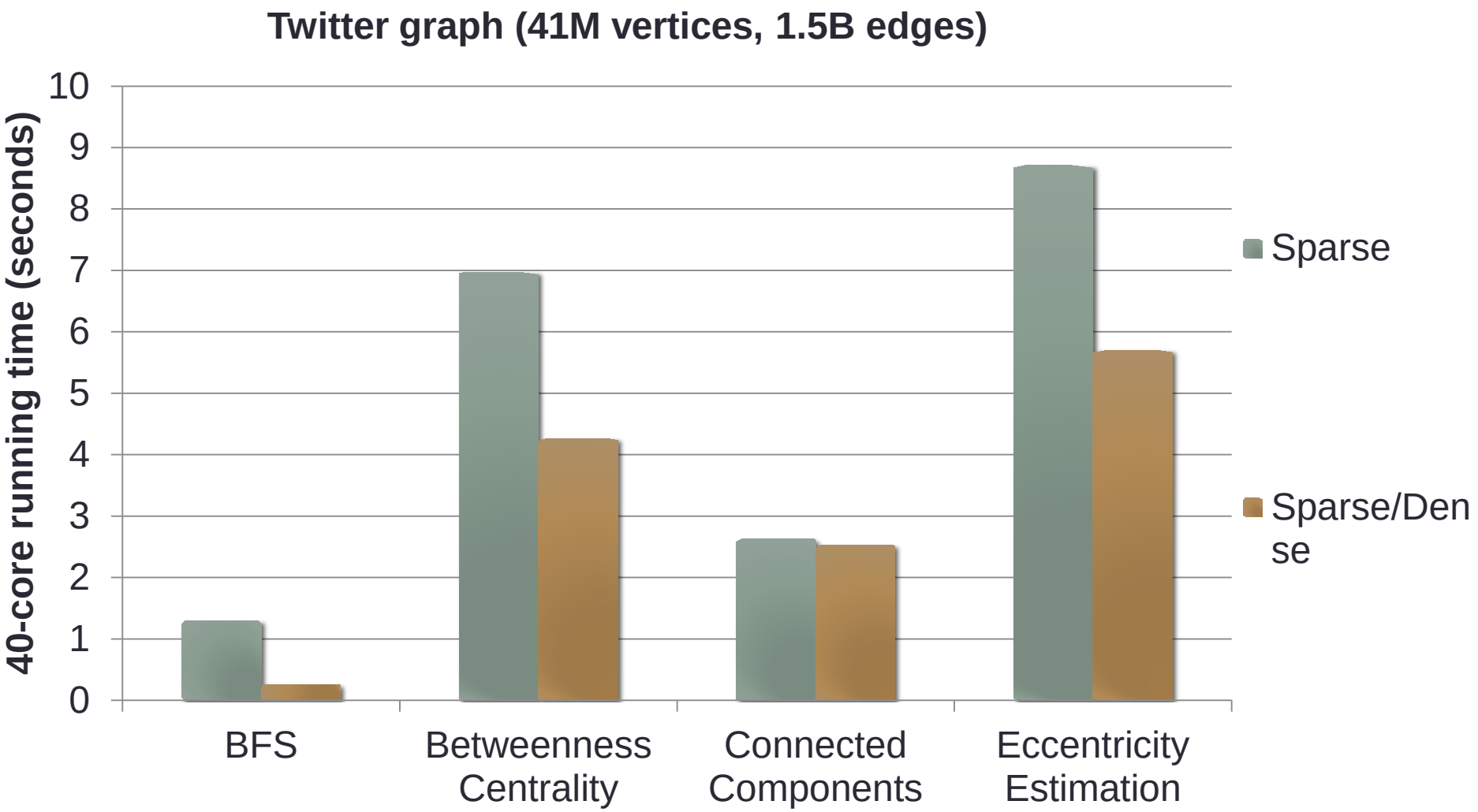


(e) PageRank-Delta

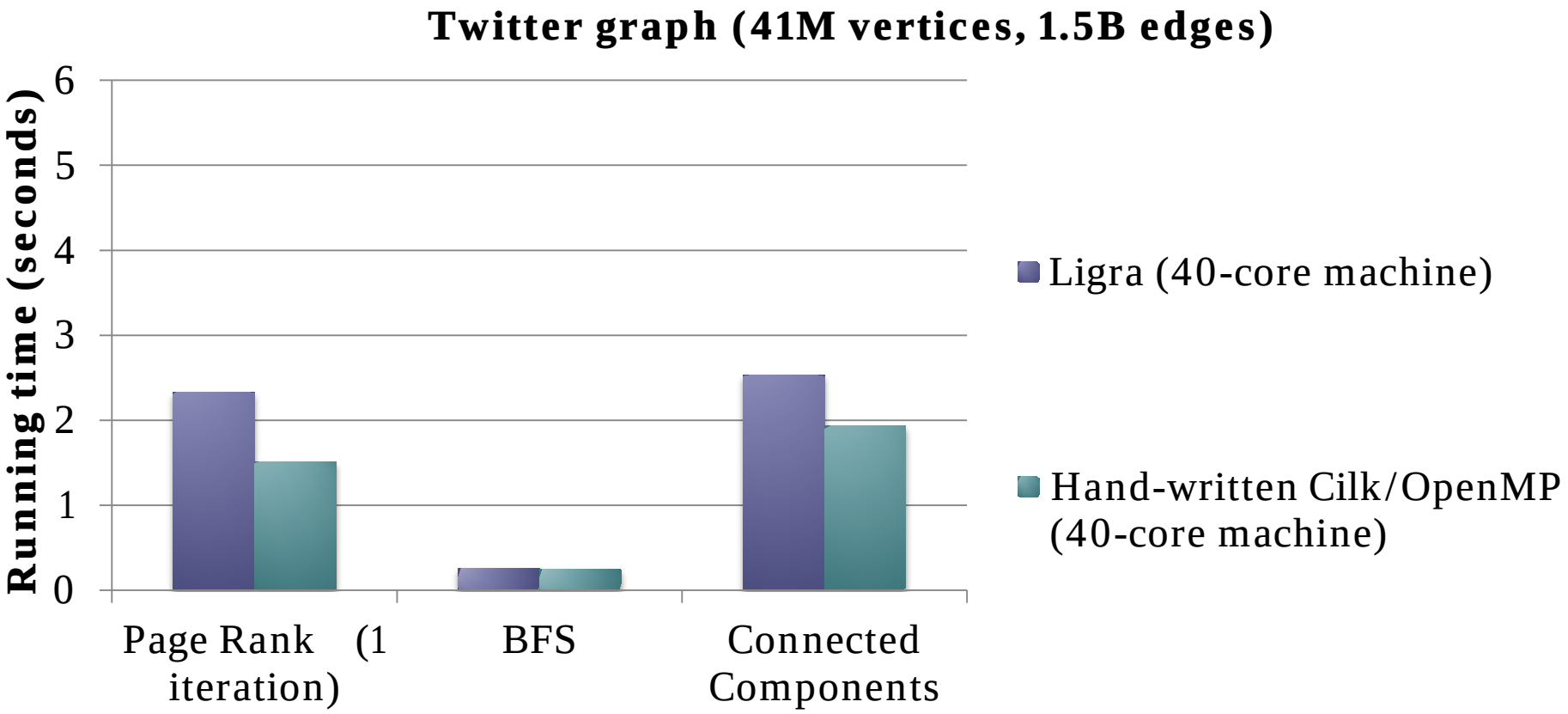


(f) Bellman-Ford

Benefit of Sparse/Dense Traversal

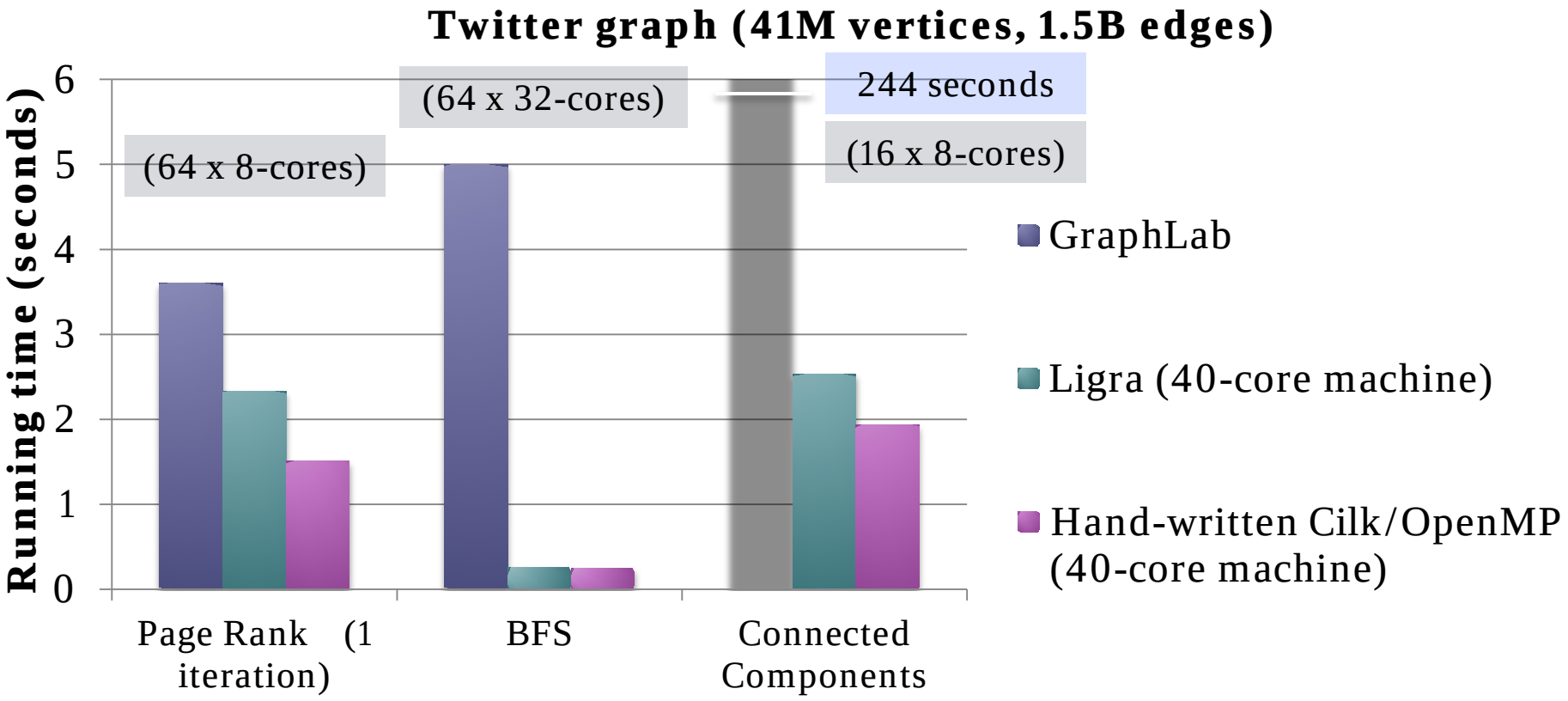


Ligra Performance



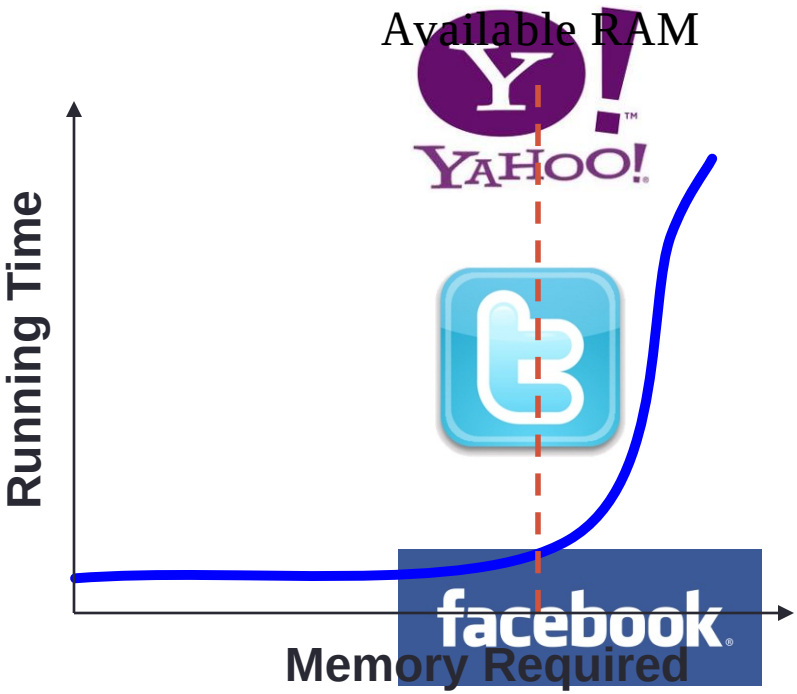
- Ligra performance close to hand-written code

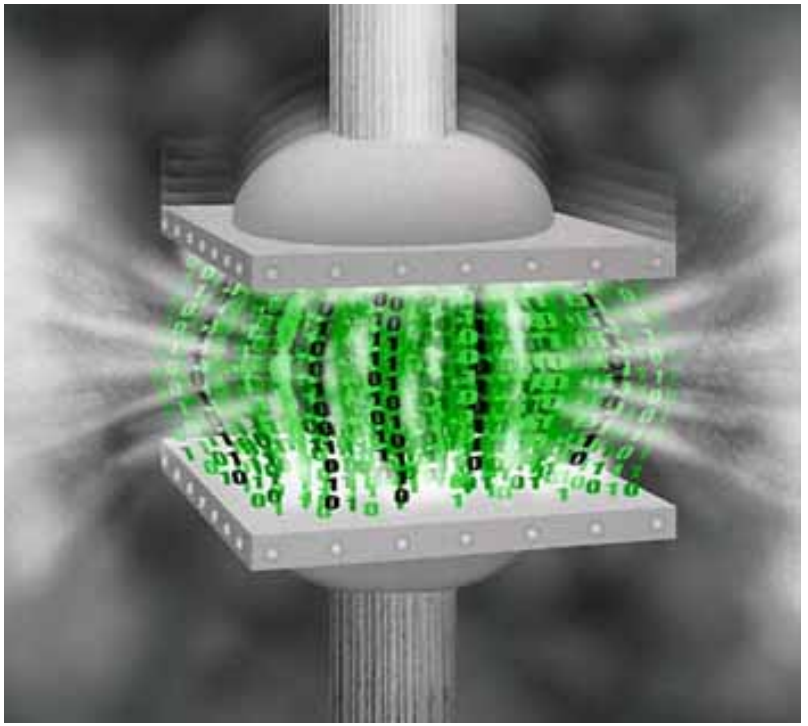
Ligra Performance



- Ligra performance close to hand-written code
- Faster than distributed-memory on per-core basis
- Several shared-memory graph processing systems subsequently developed: Galois [SOSP '13], X-stream [SOSP '13], PRISM [SPAA '14], Polymer [PPoPP '15], Ringo [SIGMOD '15]

Large Graphs





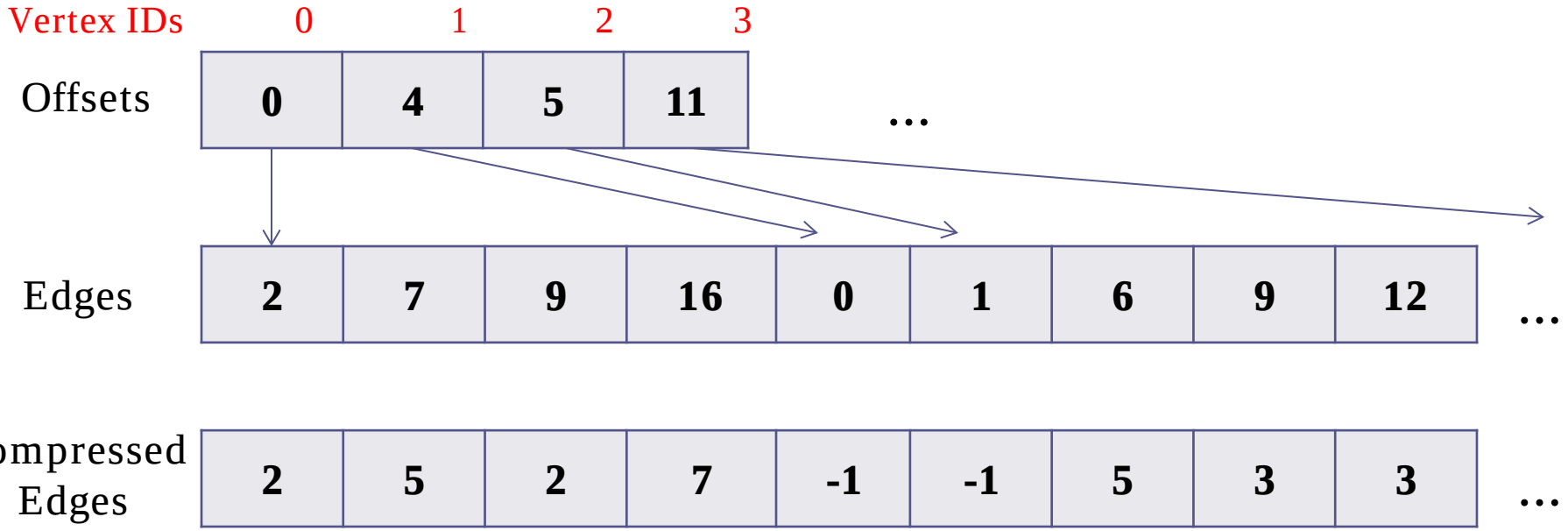
- All fit in a Terabyte of memory; can fit on commodity shared memory machine
- *What if you don't have that much RAM, or don't want to "rent" that much RAM?*

Ligra+: Adding Graph Compression

- Difference encoding (using variable-length codes) for sorted edges per vertex
- Modify EdgeMap: parallel edge decoding on-the-fly
- All hidden from the user!

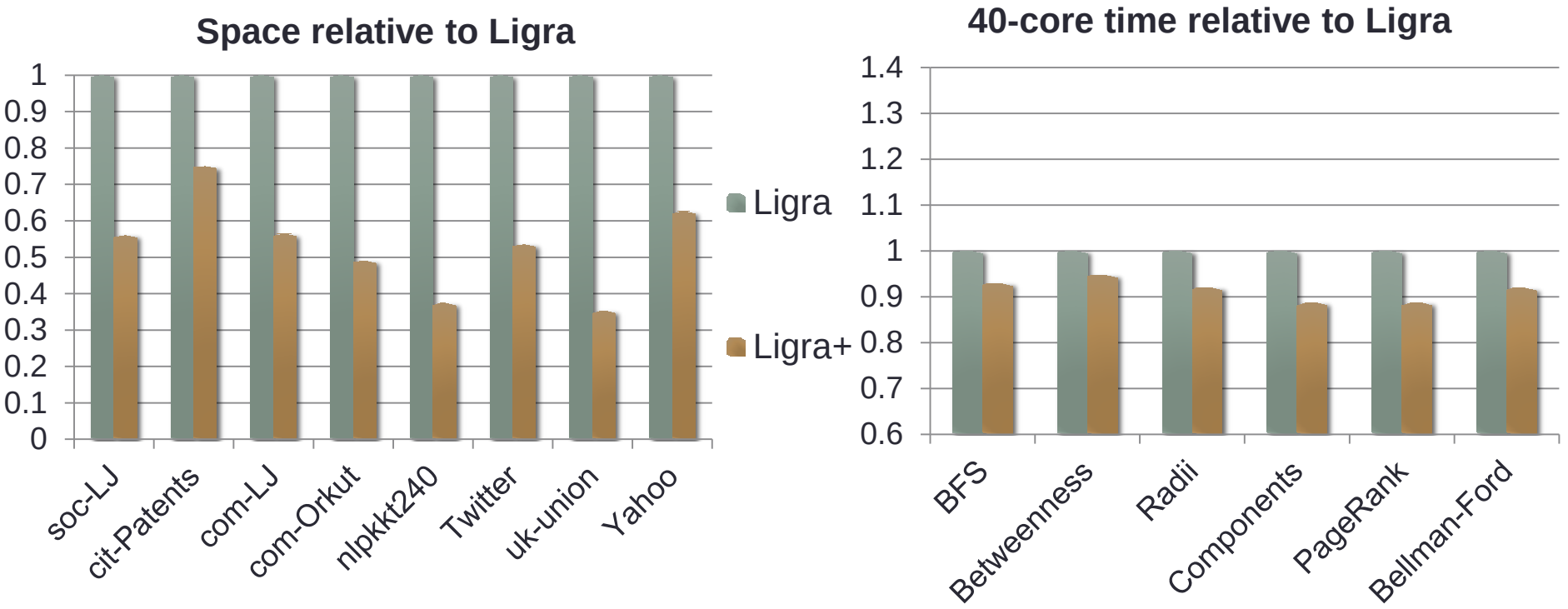
Graph Compression

- Compress the graph so that it uses less memory



  - Sort edges
  - First edge: Store difference between source and target vert
  - Subsequent edges: store difference with previous edge
- Graph reordering to improve locality
  - Goal: give neighbors IDs close to vertex ID
  - BFS, DFS, METIS, our own separator-based algorithm

Ligra+: Adding Graph Compression



- Cost of decoding on-the-fly?
- Memory bottleneck a bigger issue as graph algorithms are memory-bound

Conclusion

- Ligra: lightweight graph processing framework for shared-memory
  - “frontier-based” algorithms
  - Switches computation based on frontier size
- Ligra+: extension which incorporates graph compression
  - Reduces space usage and improves parallel performance
- Code: http://github.com/jshun/ligra