Bounded Asynchrony and Nested Parallelism for Scalable Graph Processing

Adam Fidel

Nancy Amato and Lawrence Rauchwerger

Parasol Laboratory
Department of Computer Science and Engineering
Texas A&M University

Parasol

Graph Processing

- Graph analytics is everywhere
  - Web, recommendation, social networks, science, intelligence

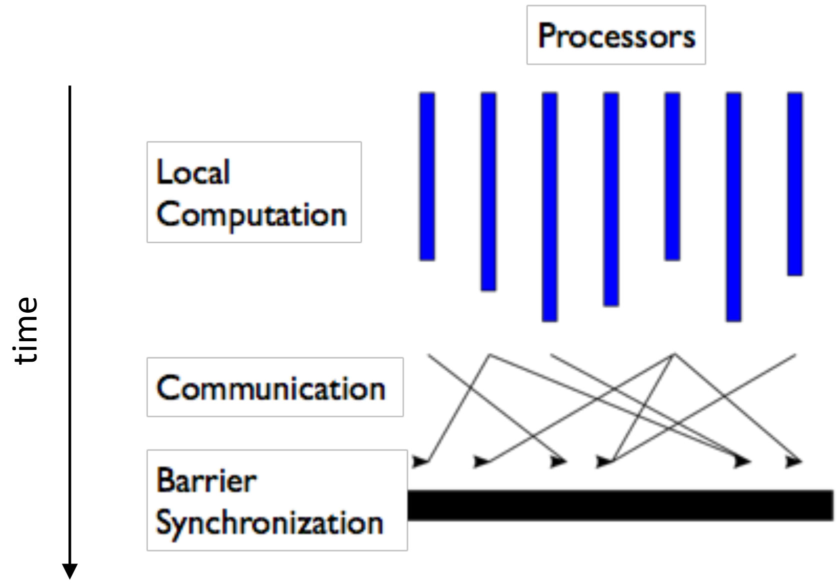
- Today's graphs of interest are extremely large
  - 100s of billions of nodes and trillions of edges
- Need for efficiently processing graphs at this scale
  - Parallel processing comes with its own set of challenges
  - Giraph, GraphLab, PowerGraph, GraphX, Galois, Green-Marl
  - STAPL Graph Library (SGL)
  - https://gitlab.com/parasol-lab/stapl

Bounded Asynchrony

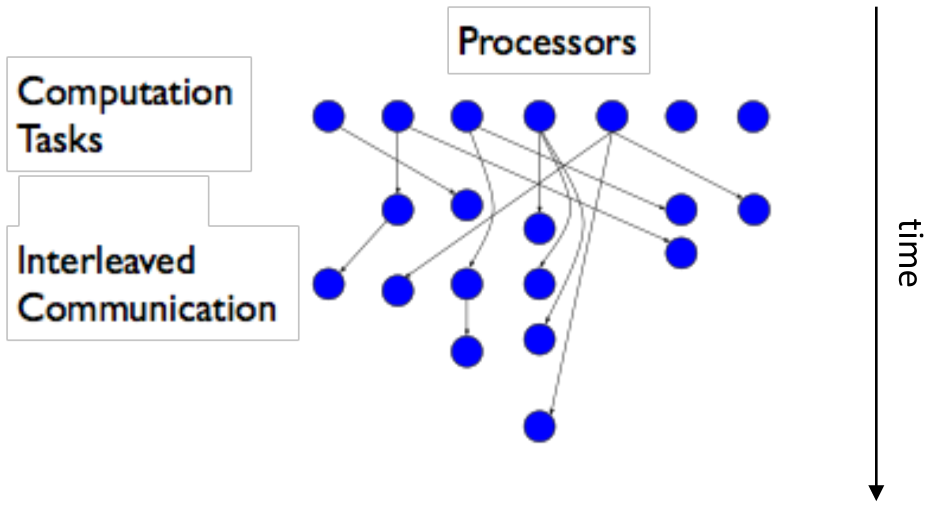
The k-level Asynchronous Model

Parallel Graph Algorithms

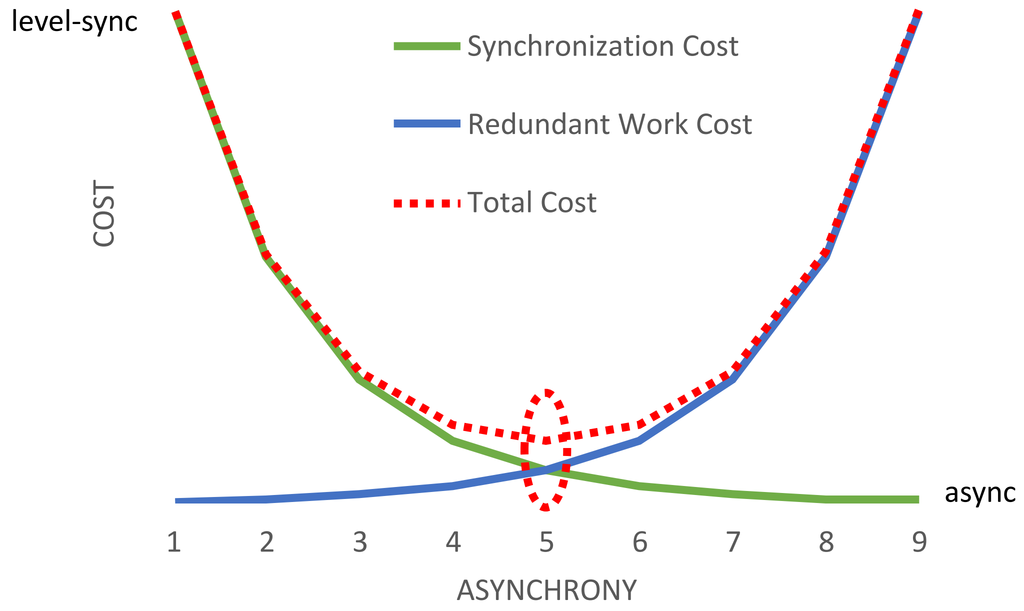
- Level-Synchronous Approach
  - BSP-model iterative computation
  - Global synchronization after each level, no redundant work



- Asynchronous Approach
  - Asynchronous task execution
  - Point-to-point synchronizations, possible redundant work



k-level-asynchronous Paradigm



- Unifies level-synchronous and asynchronous
- *k* defines depth of superstep (KLA superstep)
  - *k* = 1: level-synchronous
  - *k* = diameter: asynchronous

SGL Programming Model

KLA Breadth-First Search

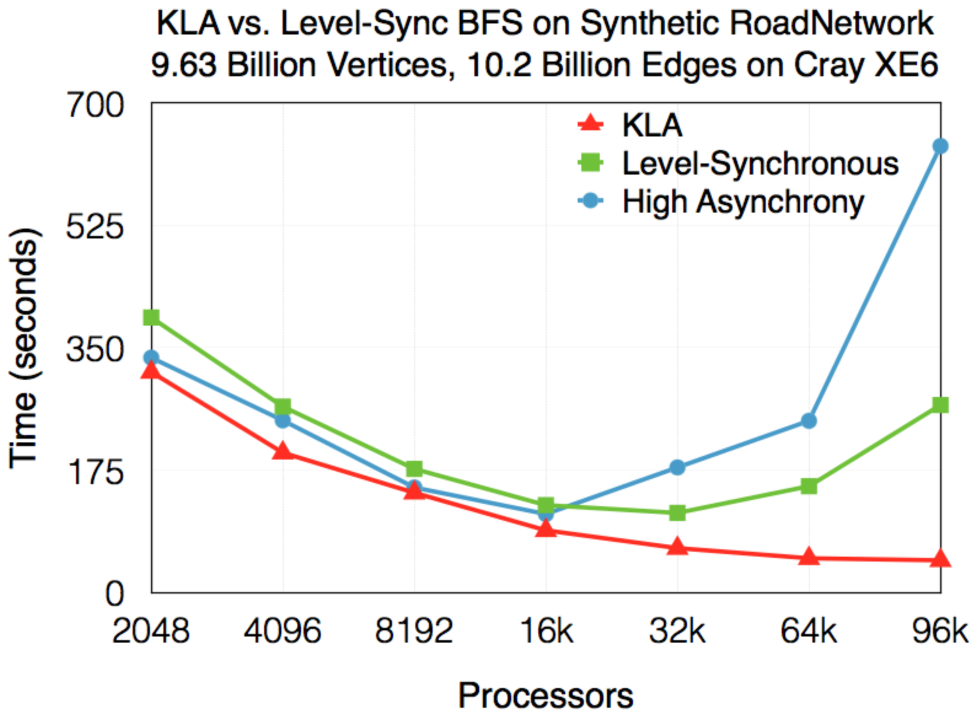
```
Function VertexOperator(v)
 if v.color = GREY then
    v.color = BLACK
    VisitAllNeighbors(v, NeighborOp,
                      v.dist+1, v.id)
    return true
 else
    return false
```

(a) Process a vertex and issue neighbor visits

```
Function NeighborOp(u, dist, parent)
 if u.dist > dist then
    u.dist ← dist
    u.parent ← parent
    u.color ← GREY
    return true
 else
    return false
```

(b) Process a neighbor

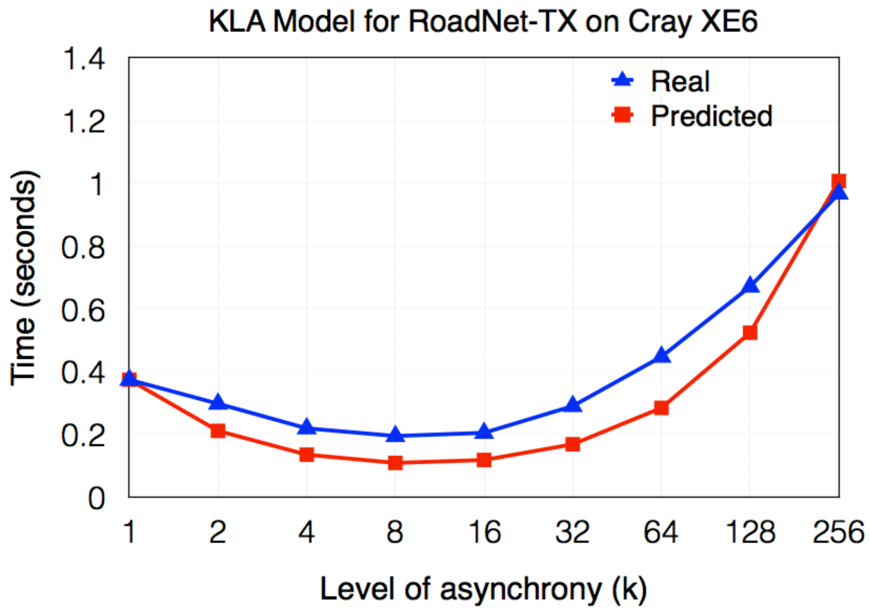
Scalability of KLA BFS

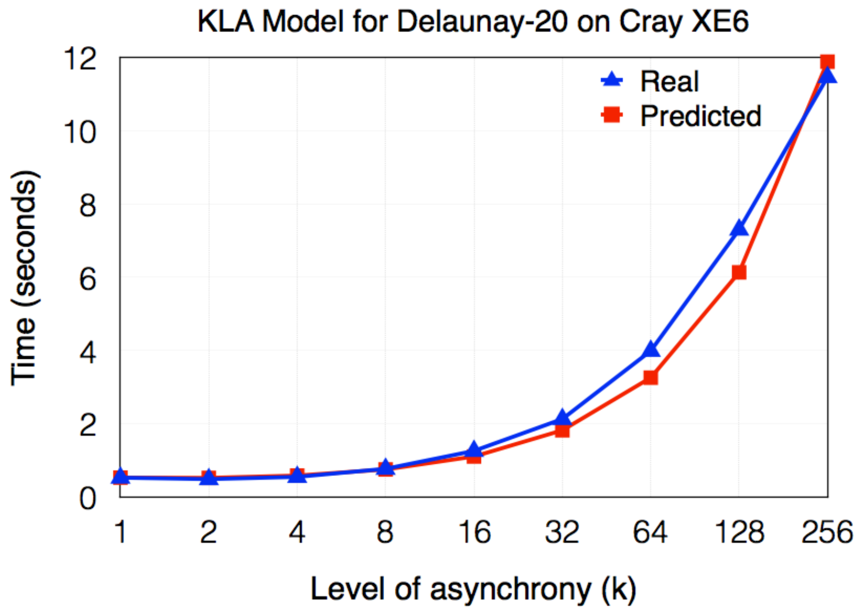


- Current strategies stop scaling after 32,768 cores
- KLA strategy faster, scales better
- Adaptively change asynchrony to balance global-synchronization costs and asynchronous penalty

Choosing k

- The level of asynchrony (k) is problem instance specific

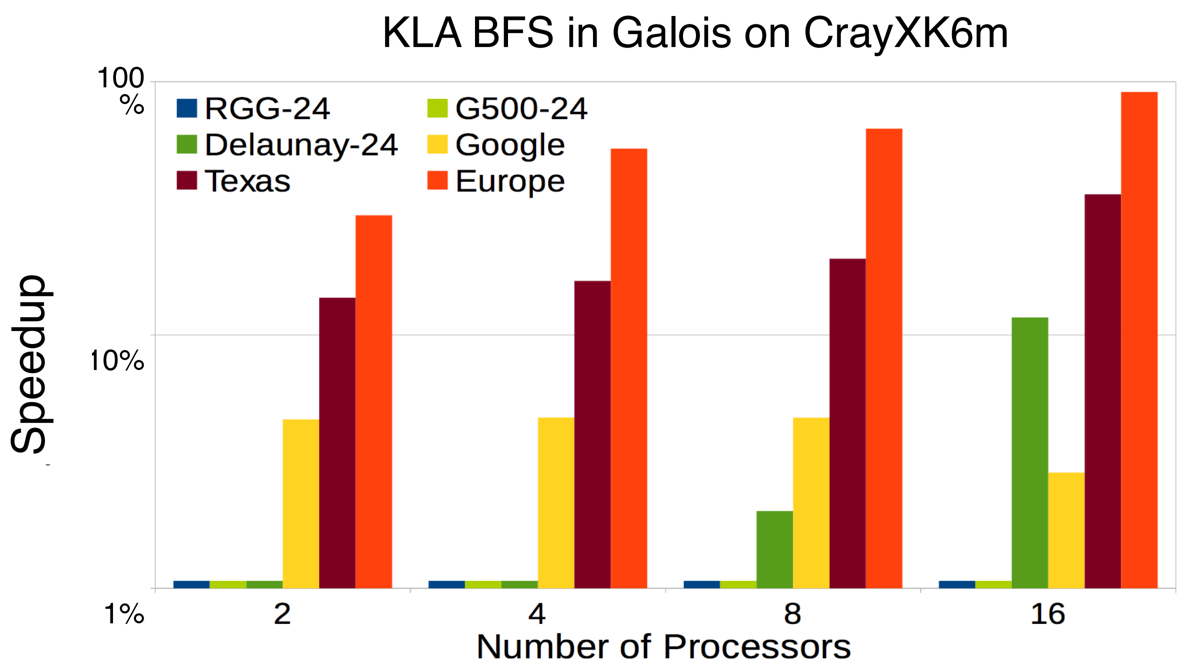




- Model the execution time for a given k
- In practice, we provide an adaptive selection method for k

KLA in Other Frameworks

- 16 cores on a single Cray XK6m node
- Modified Level-Synchronous worklist for Galois to allow for KLA
- Improvement dependent on graph type
- Performance improves vs. level-sync and async executions



Approximation Through Asynchrony

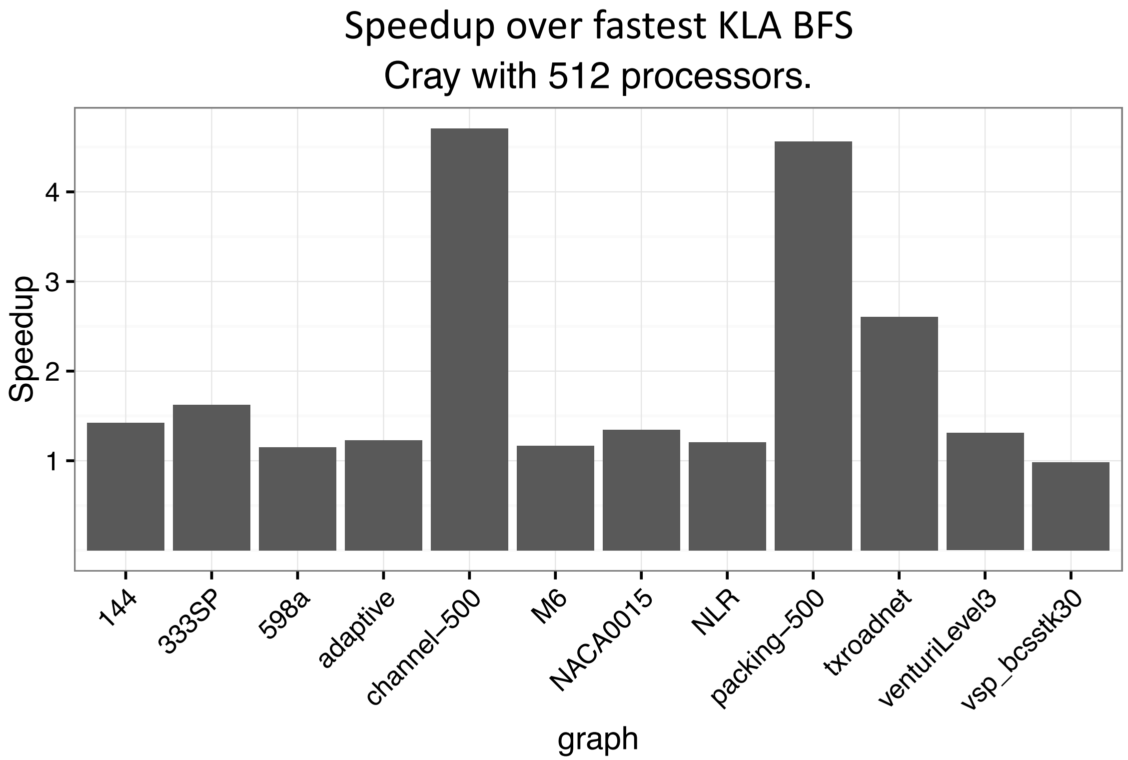
Case Study with Breadth-first Search

Parasol Distance Queries

- Shortest path between pair of vertices in a graph
- Many important applications for graphs
  - Distances in road networks, connections in social networks
- Use parallel and distributed algorithms
- Use **approximation** to reduce work (and execution time)
- Example: approximate distances between vertices
  - Speed up applications that can tolerate error
  - Unweighted graphs, parallel breadth first search

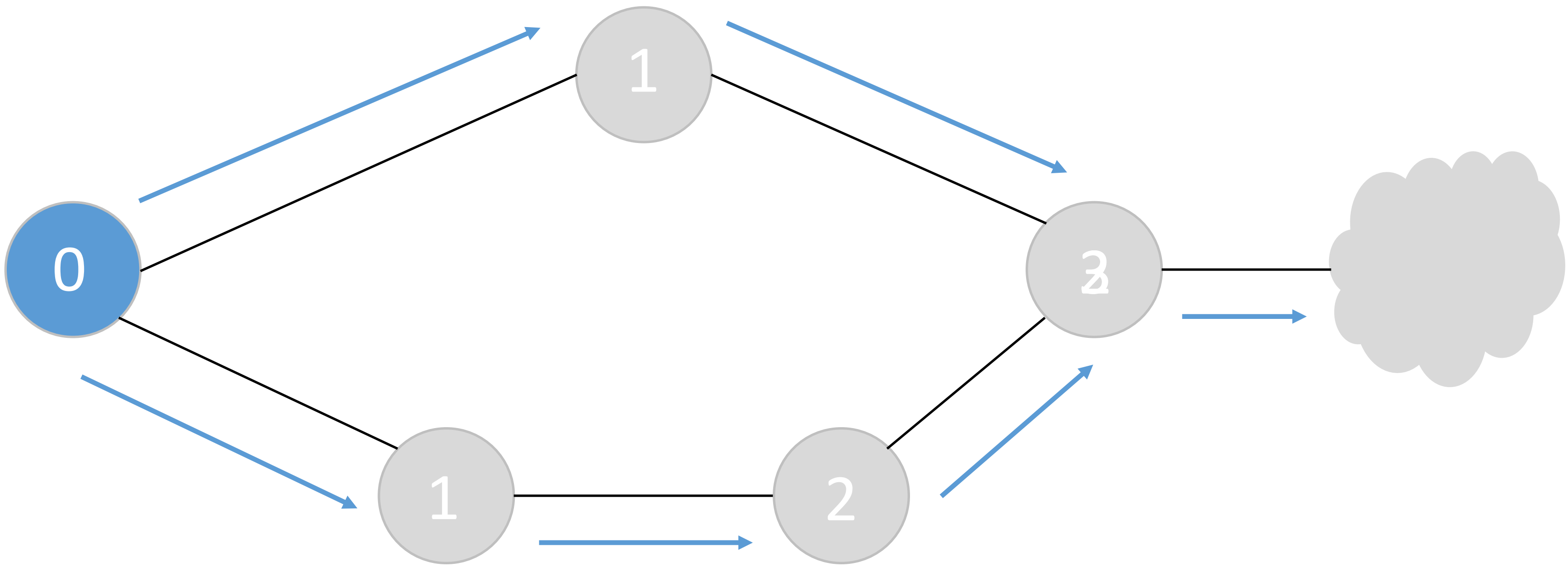
Parasol Parallel Approximate Breadth-First Search

- A new asynchronous approximate breadth-first search algorithm
- Increase asynchrony (parallelism) by reducing redundant work
- Based on the k-level-asynchronous paradigm



Where is the redundant work?

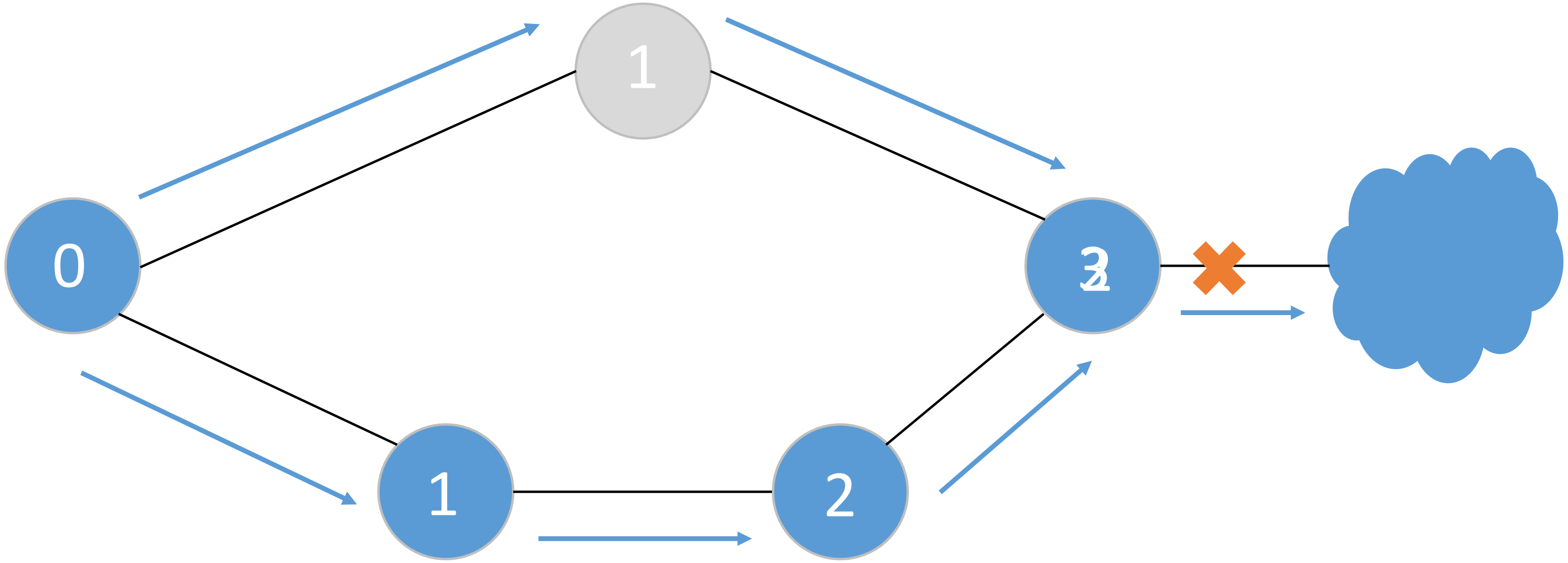
Example



Parasol

Avoiding Redundant Work

Approximate Distance

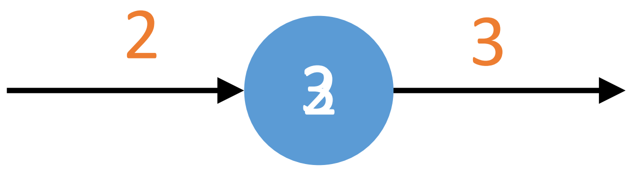


Approximate Distances

KLA Approximate Breadth-First Search

- If better distance d_{new} arrives, only propagate if sufficiently better than current distance d
  - Define a tolerance $0 \leq \tau < 1$

- **Propagate new distance if $(d - d_{new})/d \geq \tau$**



$\tau = 0.4$

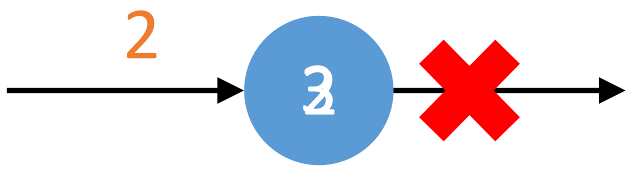
$(3-2)/2 \geq 0.4$

Approximate Distances

KLA Approximate Breadth-First Search

- If better distance d_{new} arrives, only propagate if sufficiently better than current distance d
  - Define a tolerance $0 \leq \tau < 1$

- **Propagate new distance if $(d - d_{new})/d \geq \tau$**



$\tau = 0.6$

(3-2) 0.6 ?

Approximate BFS

```
Function ApproxNeighOp(u, dist, par)
  if u.dist > dist then
    u.dist ← dist
    if (u.prop - dist)/u.prop ≥ τ then
      u.parent ← par
      u.color ← GREY
      u.prop ← dist
      return true
  else
    return false
```

(a) Approximate

```
Function NeighborOp(u, dist, par)
  if u.dist > dist then
    u.dist ← dist


    u.parent ← par
    u.color ← GREY

    return true
  else
    return false
```

(b) Original

Error Bounds

- d(v): Exact distance from source
- $d^{\tau}_{k}(v)$: Distance found with approximate algorithm

- At the end of the algorithm, all reachable vertices will have distance **$d^{\tau}_{k}(v) \leq k \times d(v)$**

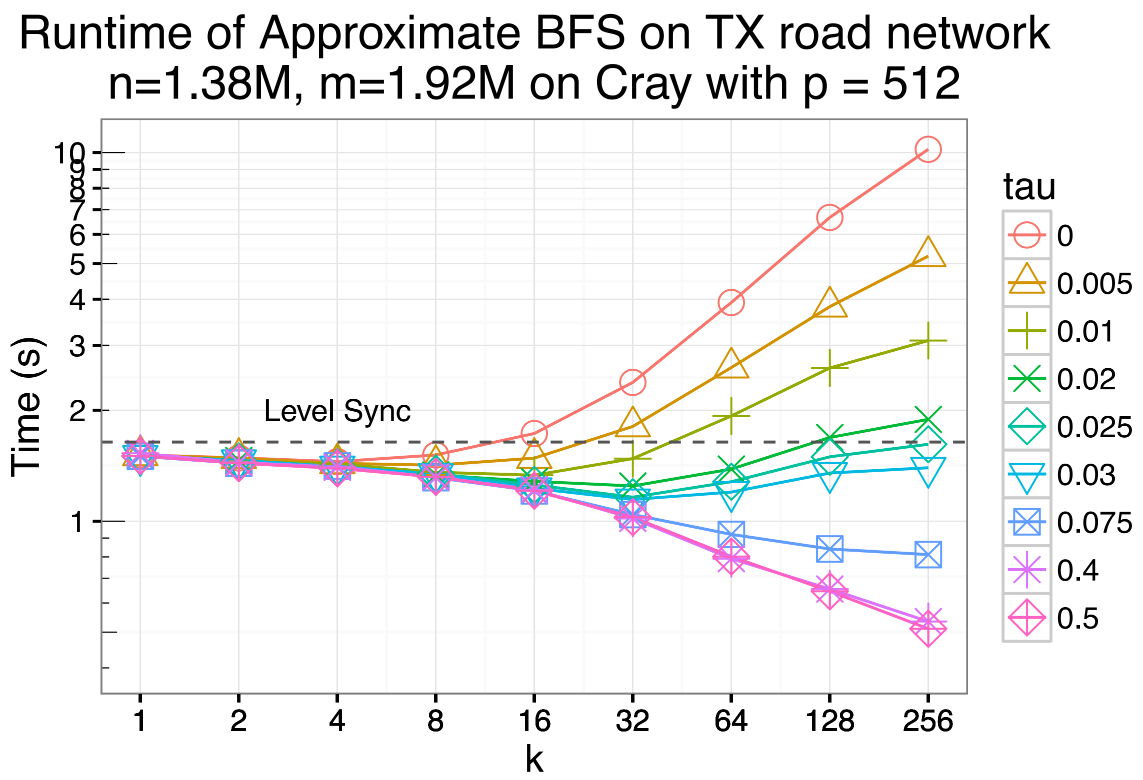
- Proof in the paper

Parasol

Experimental Setup

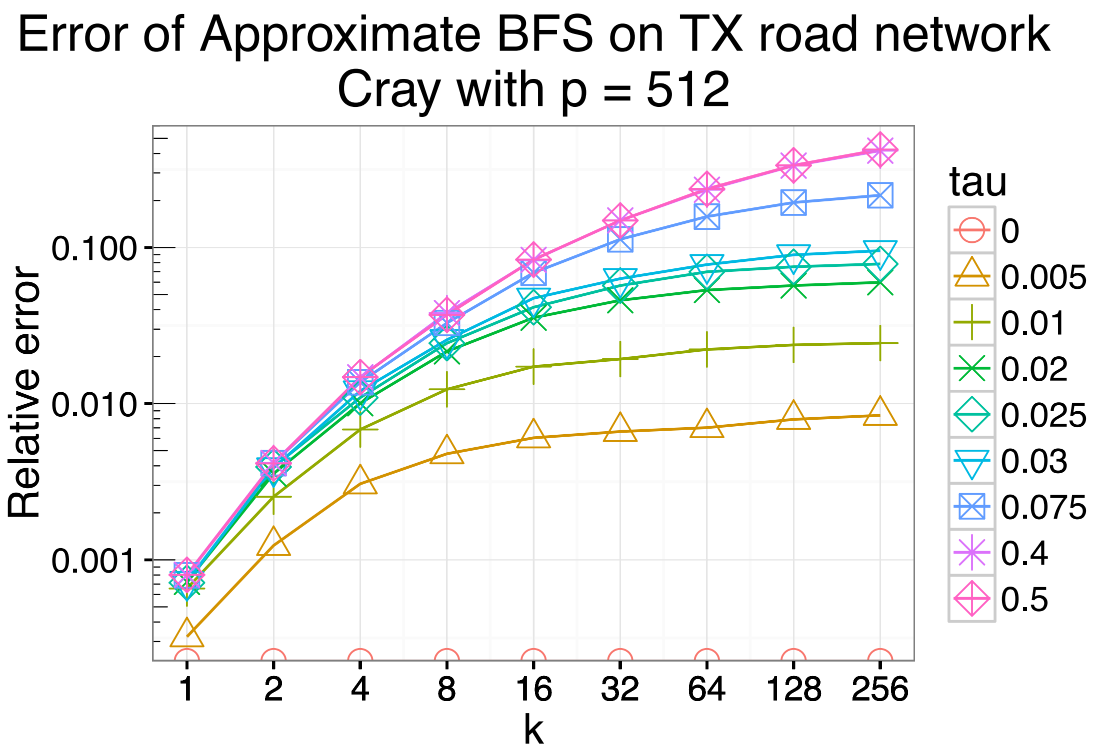
- Implemented in STAPL using STAPL Graph Library
- Cray-XK7 (TAMU)
  - 24 nodes of 16-core AMD Interlagos processors
    - 12 single socket and 12 dual socket nodes.
- IBM-BG/Q (LLNL)
  - 24,576 nodes, each node with a 16-core IBM PowerPC A2 processor
- Experiments are mean of 32 trials, with 95% confidence intervals

Texas Road Network (performance)



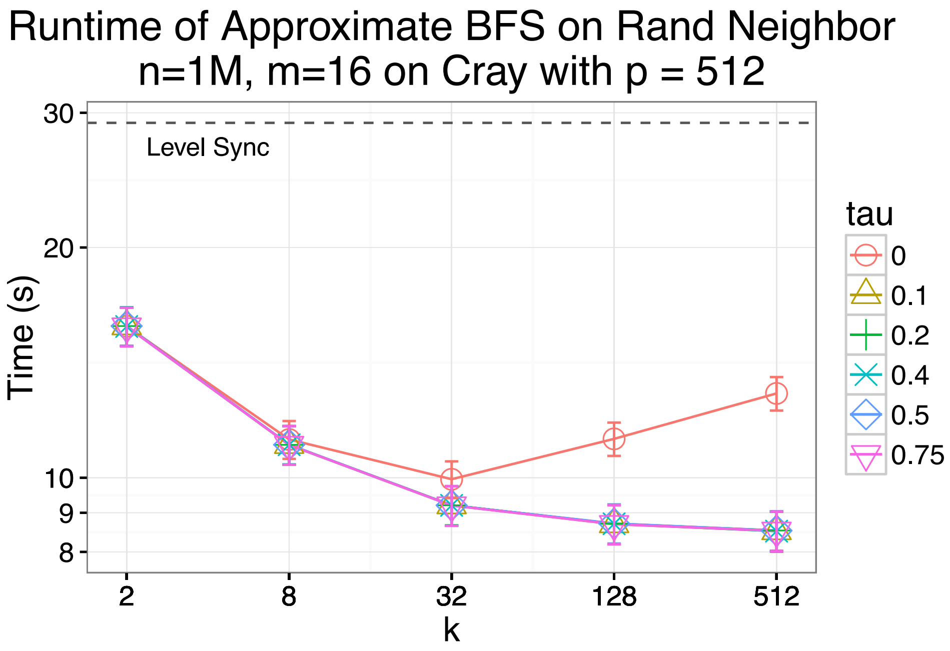
- Exact algorithm (τ = 0) is worse with higher asynchrony
- High τ (0.5) is faster with higher asynchrony (2.6x), but with error…

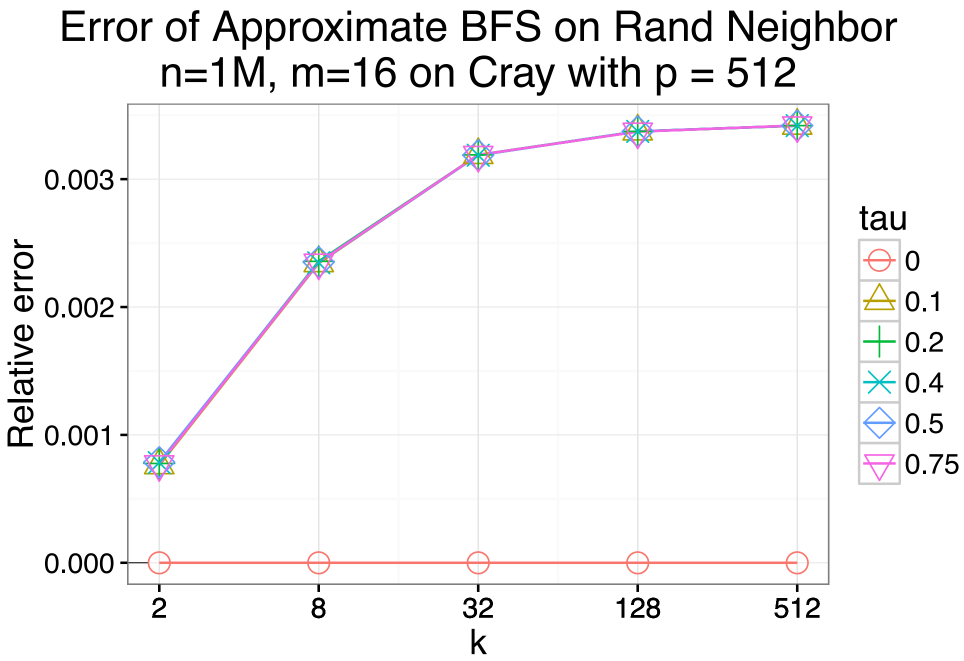
Texas Road Network (error)



- Relative error for distance of a vertex $(d^{\tau}_{k}(v) - d(v))/d(v)$
  - Shown is mean of all vertices
- Higher values of k and τ lead to higher error

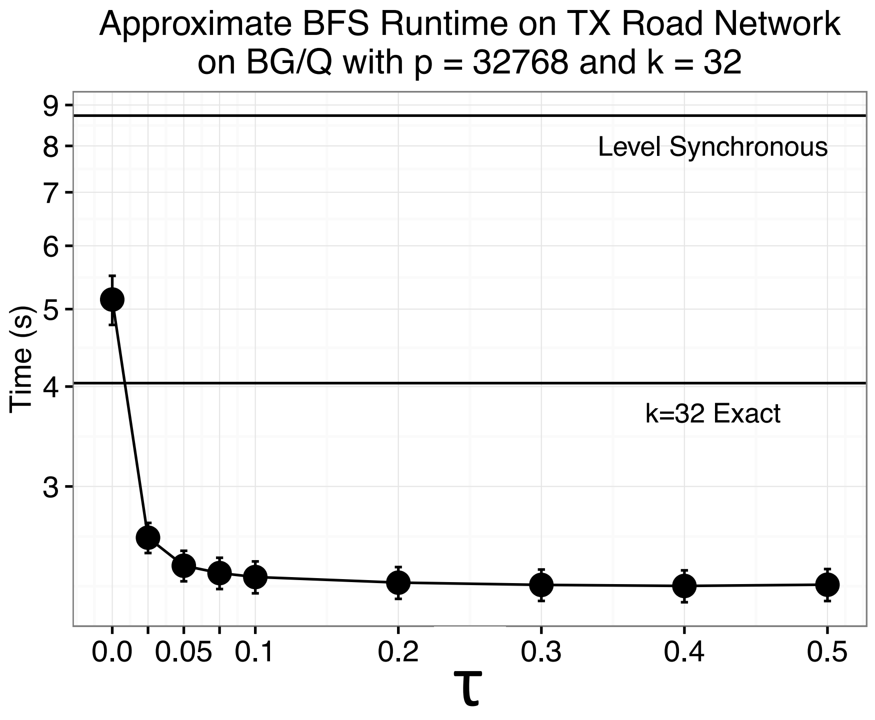
Random neighborhood

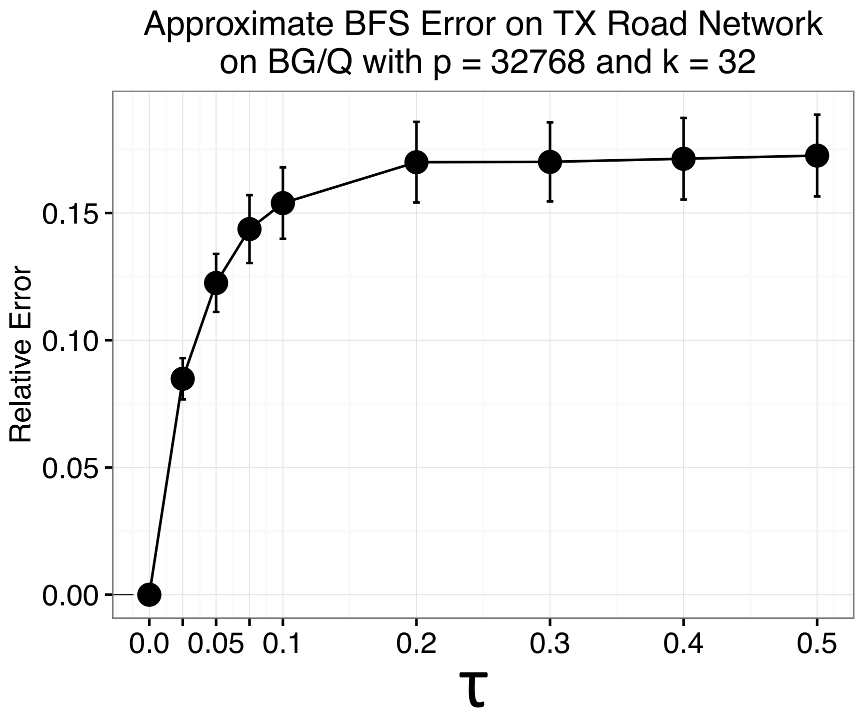




- Lower speedup (1.12x), but lower error

Texas Road Network (BG/Q)





- Fixed value of k, varying τ
- Higher τ gives better performance, with more error

Nested Parallelism

Efficiently Processing Scale-Free Graphs

Parasol Scale-Free Graphs

- Many real-world graphs are scale-free
  - Degrees follow a power-law distribution
  - Presence of “hub” vertices connected to most of graph
- Hub vertices pose many challenges
  - Load imbalance when processing visits
  - May not fit into main memory of single machine
- Current techniques “partition” the hubs
  - Ghosting, delegates, hierarchical representation
  - Rigid partitioning and ad-hoc solution

Parasol

Our Solution

- Use nested parallelism to visit edges during traversals

- Apply different strategies for hubs and non-hubs
  - Provide several strategies for distributing the edges of hub vertices, that can be seamlessly interchanged.

- Same vertex-centric specification of the algorithm

Processing SGL graph algorithms

```
while (spawned > 0)
    spawned = 0
        for (v in V) par do
            if (v.active)
                for ((v, u) in adj(v)) do
            spawn(neighbor-op, klass, u, v.level+1)
            spawned += v.active
        klass += k
```

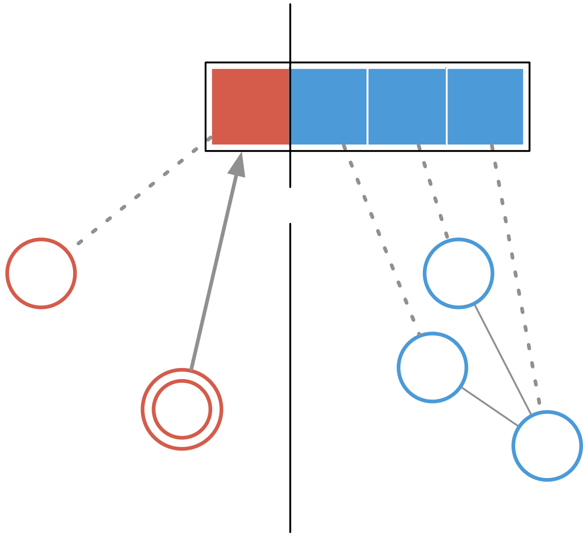
Nested Parallelism

```
while (spawned > 0)
  spawned = 0
    for (v in V) par do
        if (v.active)
            for ((v, u) in adj(v)) par do
          spawn(neighbor-op, klass, u, v.level+1)
        spawned += v.active
    klass += k
```

Nested Parallel Visitation

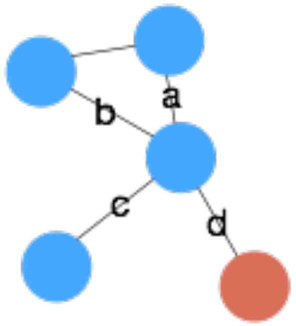
```
void visit_all_neighbors(v, op)
   map(resolve_neighbor, v.edges(), op)


  void resolve_neighbor(e, op)
     spawn(op, e.target)
```



- We perform a map (parallel for all) inside of the vertex processing
- For distributed vertices, nested parallel algorithm executes on locations that store edges for v

Adjacency List Partitions

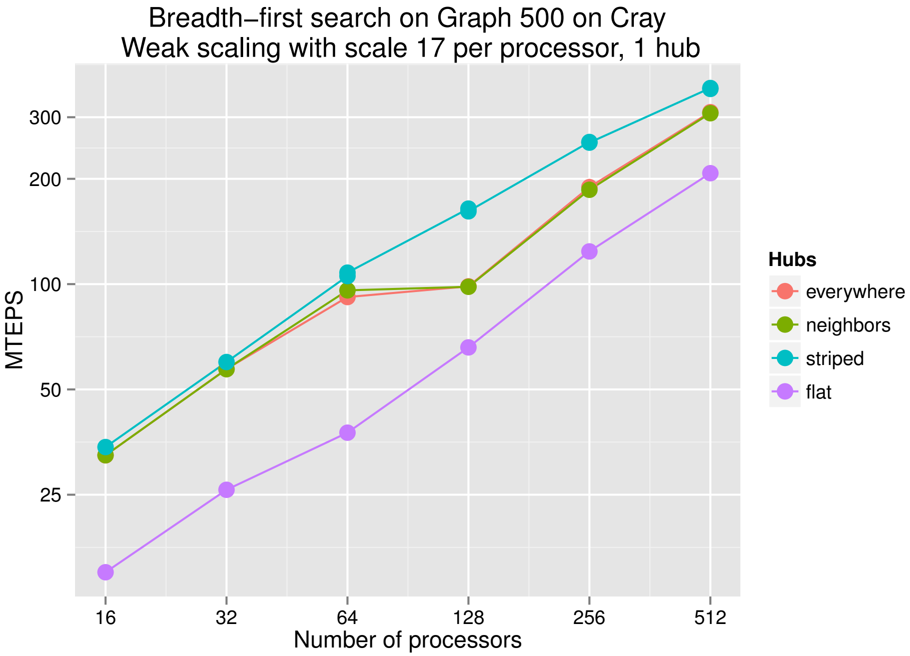


- Randomized-Balanced
  - Use the same set of locations of graph
  - Create a balanced partition across those locations

- Neighbors
  - Use the same set of locations of graph
  - Place an edge (s,t) on the same location as t

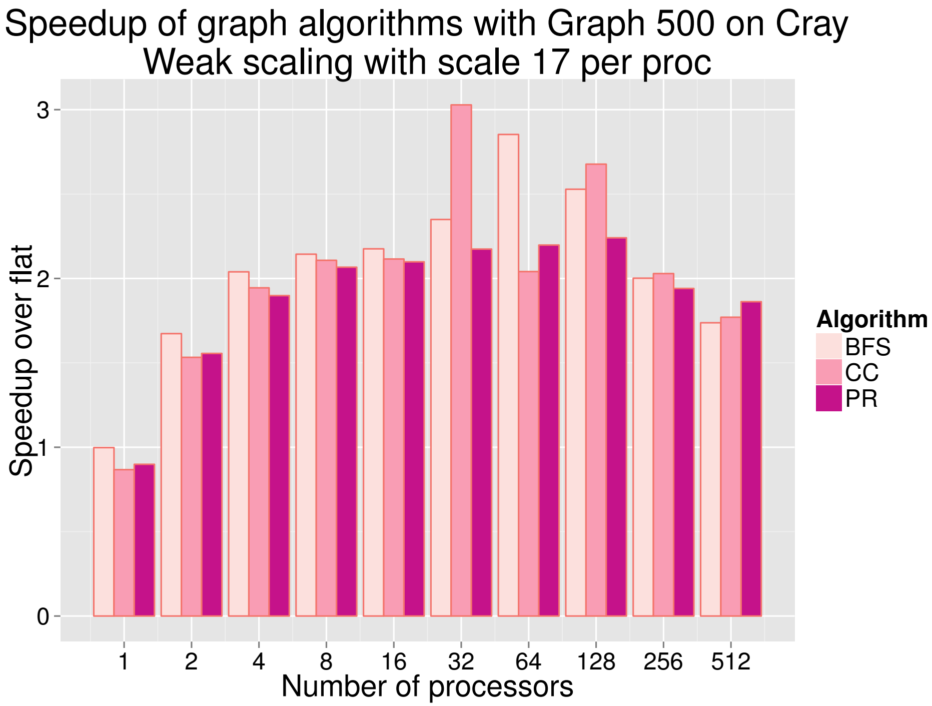
- Hierarchical
  - Use only one location per shared-memory node
  - Create a balanced partition across those locations

Breadth First Search



- Hubs are chosen by selecting top vertices based on degree
- All strategies are faster than flat
- Hierarchical is significantly faster than the others on Cray

Other Algorithms



- Other algorithms besides BFS
- Speedup is T_{flat} / T_{oracle} where T_{oracle} is the fastest nested configuration

Conclusion

- Bounded asynchrony can increase performance of graph algorithms
- Asynchrony can be used to tradeoff error for performance
- Nested parallelism boosts performance of algorithms in the presence of hubs