Parallel Local Graph Clustering

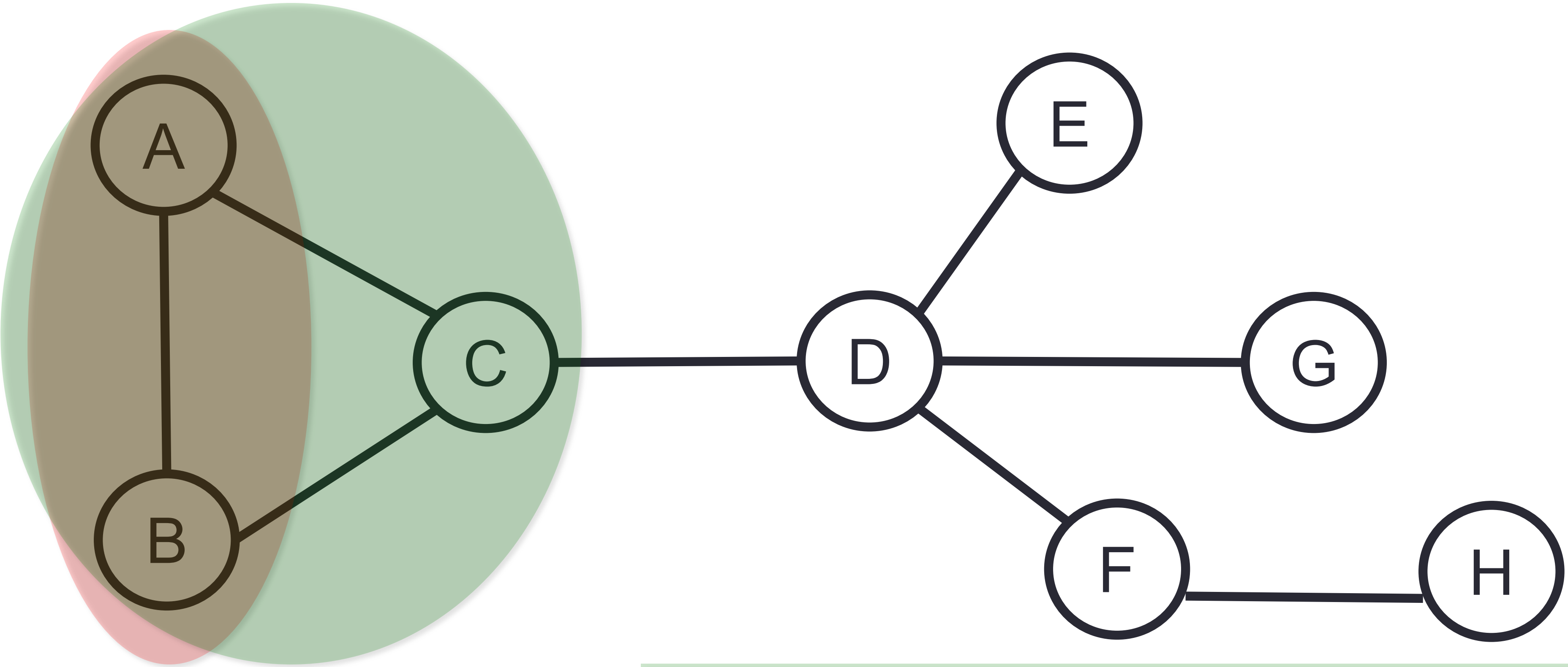
Julian Shun

Joint work with Farbod Roosta-Khorasani, Kimon Fountoulakis, and Michael W. Mahoney

Work appeared in VLDB 2016

Metric for Cluster Quality

$$\text{Conductance} = \frac{\text{Number of edges leaving cluster}}{\text{Sum of degrees of vertices in cluster*}}$$



Conductance = 2/(2+2) = 0.5

Conductance = 1/(2+2+3) = 0.14

Low conductance → “better” cluster

*Consider the smaller of the two sides

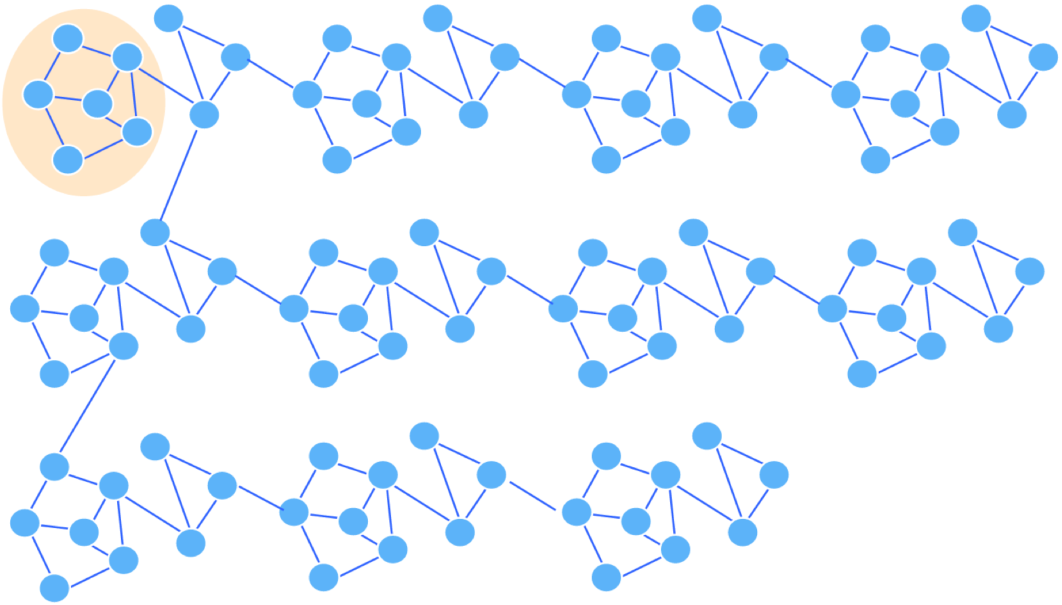
Clustering Algorithms

- Finding minimum conductance cluster is NP-hard
- Many approximation algorithms and heuristic algorithms exist
  - Spectral partitioning, METIS (recursive bisection), maximum flow-based algorithms, etc.
- All algorithms are global, i.e., they need to touch the whole graph at least once requiring at least |V|+|E| work
  - Can be very expensive for billion-scale graphs

YAHOO!	Common Crawl	facebook
1.4 billion vertices 6.6 billion edges	3.5 billion vertices 128 billion edges	1.4 billion vertices 1 trillion edges

Local Clustering Algorithms

- Does work proportional to only the size of the output cluster (can be much less than |V|+|E|)

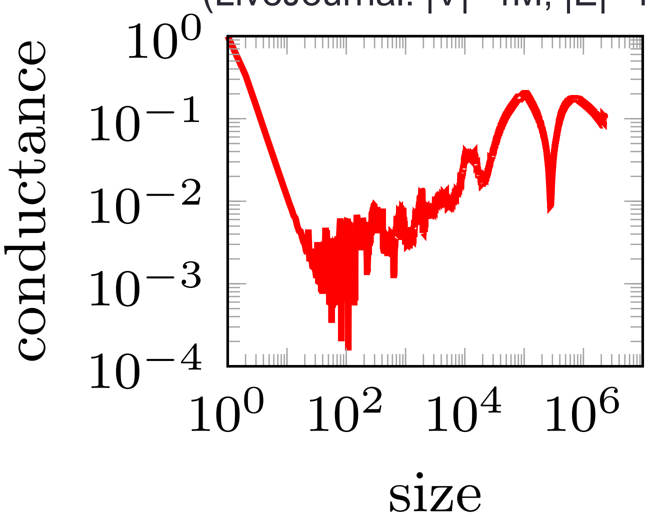


- Take as input a “seed” set of vertices and find good cluster close to “seed” set

Local Clustering Algorithms

- Many meaningful clusters in real-world networks are relatively small [Leskovec et al. 2008, 2010, Jeub et al. 2015]

Network community profile plot
(LiveJournal: |V|=4M, |E|=40M)



Some existing local algorithms

Spielman and Teng 2004
Andersen, Chung, and Lang 2006
Andersen and Peres 2009
Gharan and Trevisan 2012
Kloster and Gleich 2014
Chung and Simpson 2015

All existing local algorithms are sequential

Existing studies are on small to medium graphs

Goal: Develop parallel local clustering algorithms that scale to massive graphs

Parallel Local Algorithms

- We present first parallel algorithms for local graph clustering
  - Nibble [Spielman and Teng 2004]
  - PageRank-Nibble [Andersen, Chung, and Lang 2006]
  - Deterministic HeatKernel-PageRank [Kloster and Gleich 2014]
  - Randomized HeatKernel-PageRank [Chung and Simpson 2015]
  - Sweep cut
- All local algorithms take various input parameters that affect output cluster
  - Parallel Method 1: Try many different parameters independently in parallel
  - Parallel Method 2: Parallelize algorithm for individual run
    - Useful for **interactive** setting where data analyst tweaks parameters based on previous results

PageRank-Nibble [Andersen, Chung, and Lang 2006]

Note: $|\mathbf{p}|_1 + |\mathbf{r}|_1 = 1.0$ (i.e., is a probability distribution)

<u>Algorithm Idea</u>

Iteratively spread probability mass around the graph

- Initialize $\mathbf{p} = \{\}$ (contains 0.0 everywhere implicitly)
  $\mathbf{r} = \{(\mathbf{s},1.0)\}$ (contains 0.0 everywhere except $\mathbf{s}$)
- While(any vertex u satisfies $\mathbf{r}[u]/\deg(u) \geq \varepsilon$)
  1) Choose any vertex u where $\mathbf{r}[u]/\deg(u) \geq \varepsilon$
  2) $\mathbf{p}[u] = \mathbf{p}[u] + \alpha\mathbf{r}[u]$
  3) For all neighbors ngh of u: $\mathbf{r}[ngh] = \mathbf{r}[ngh] + (1-\alpha)\mathbf{r}[u]/(2\deg(u))$
  4) $\mathbf{r}[u] = (1-\alpha)\mathbf{r}[u]/2$
- Apply sweep cut rounding on $\mathbf{p}$ to obtain cluster

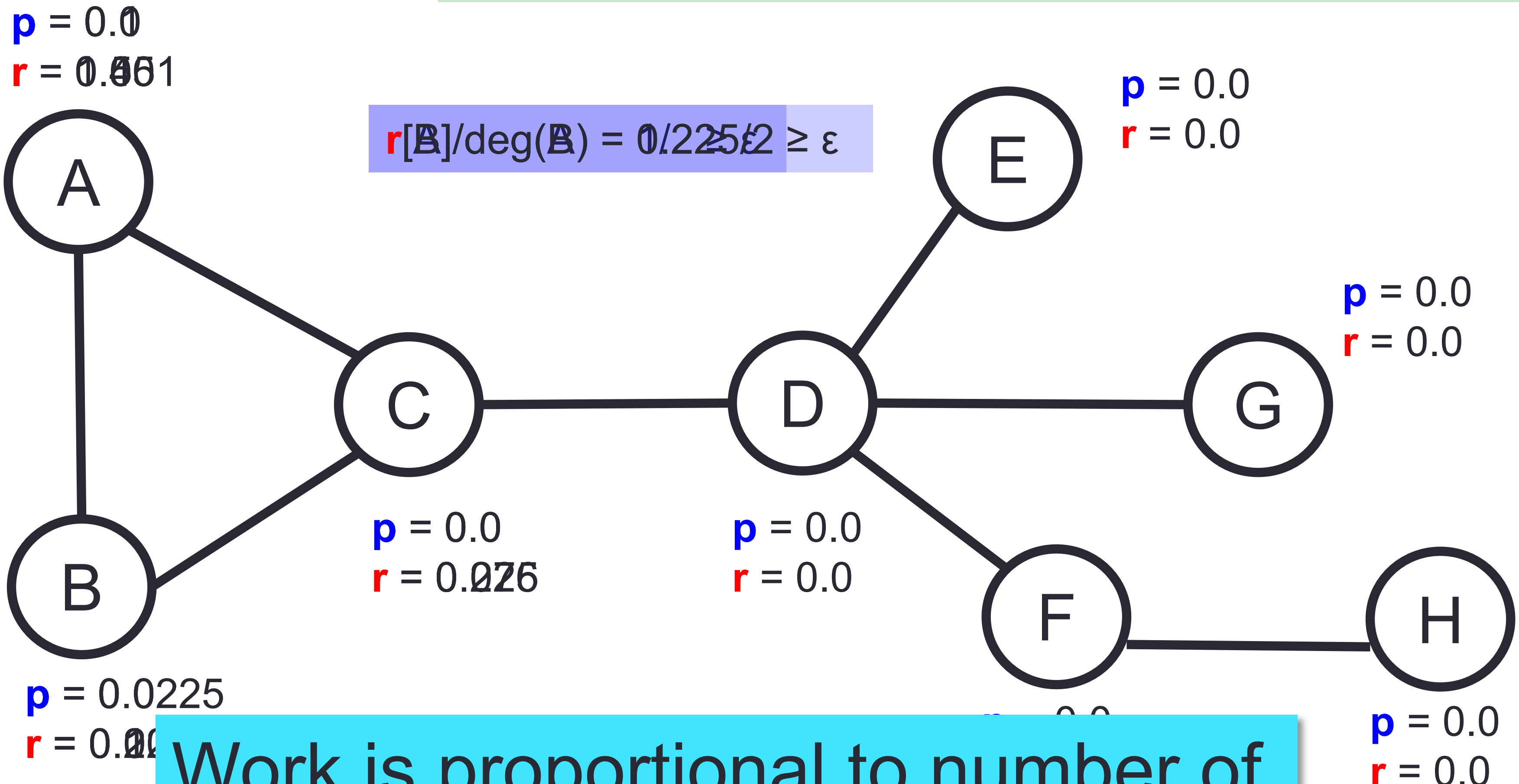
PR-Nibble

While(any vertex u satisfies **r**[u]/deg(u) ≥ ε)

1) Choose any vertex u where **r**[u]/deg(u) ≥ ε
2) **p**[u] = **p**[u] + α**r**[u]
3) For all neighbors ngh of u: **r**[ngh] = **r**[ngh] + (1-α)**r**[u]/(2deg(u))
4) **r**[u] = (1-α)**r**[u]/2

r[B]/deg(B) = 0.225/2 ≥ ε

r[A]/deg(A) = 1/2 ≥ ε



Work is proportional to number of nonzero entries and their edges

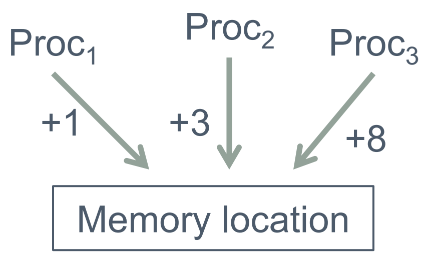
Parallel PR-Nibble

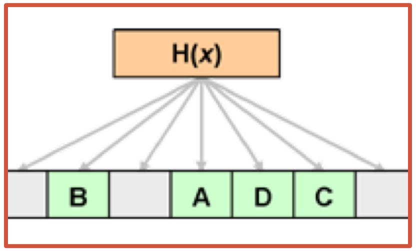
- Input: seed vertex **s**, error ε, teleportation α
- Maintain approximate PageRank vector **p** and residual vector **r** (length equal to # vertices)
- Initialize **p** = {0.0,…,0.0}, **r** = {0.0,…,0.0} and **r**[**s**] = 1.0
- While(any vertex u satisfies **r**[u]/deg(u) ≥ ε)
  1. Choose ~~any~~ ALL vertex u where **r**[u]/deg(u) ≥ ε
  2. **p**[u] = **p**[u] + α**r**[u]
  3. For all neighbors ngh of u: **r**[ngh] = **r**[ngh] + (1-α)**r**[u]/(2deg(u))
  4. **r**[u] = (1-α)**r**[u]/2
- Apply sweep cut rounding on **p** to obtain cluster

Using some stale information—is that a problem?

Parallel PR-Nibble

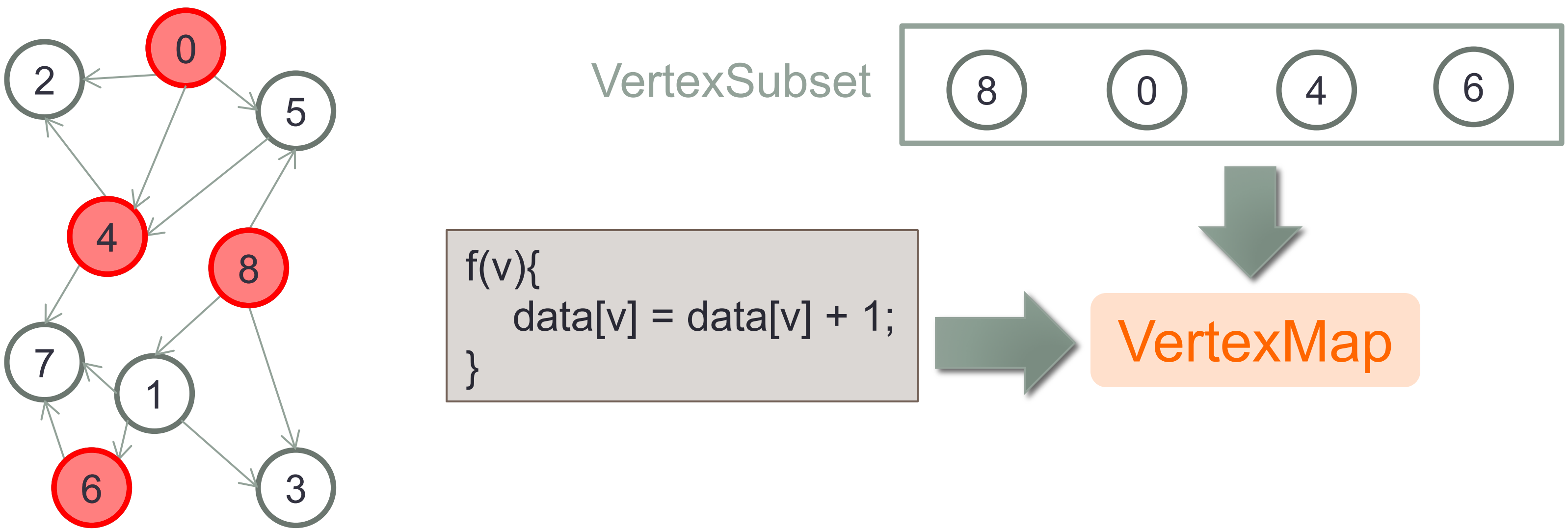
- We prove that asymptotic work remains the same as the sequential version, $O(1/(\alpha\varepsilon))$
- Guarantee on cluster quality is also maintained
- Parallel implementation:
  - Use fetch-and-add to deal with conflicts
  - Concurrent hash table to represent sparse sets (for local running time)
  - Use the Ligra graph processing framework [Shun and Blelloch 2013] to process only the "active" vertices and their edges (for local running time)





Ligra Graph Processing Framework

VertexSubset | VertexMap | EdgeMap

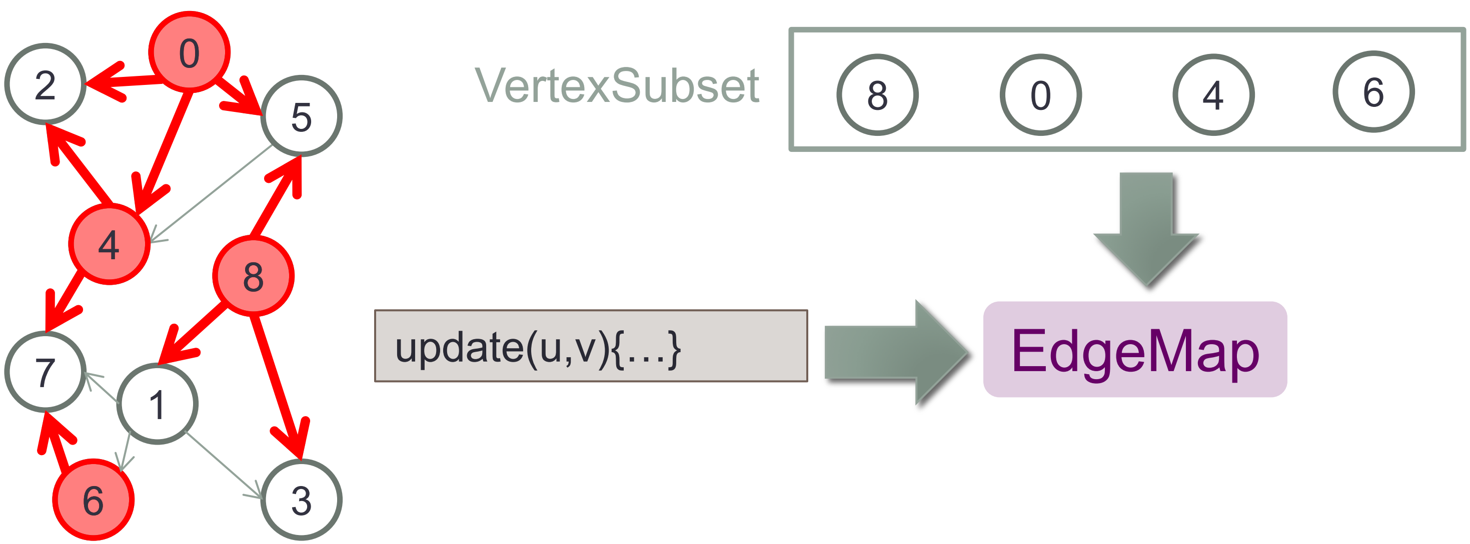


Ligra Graph Processing Framework

VertexSubset

VertexMap

EdgeMap



Parallel PR-Nibble in Ligra

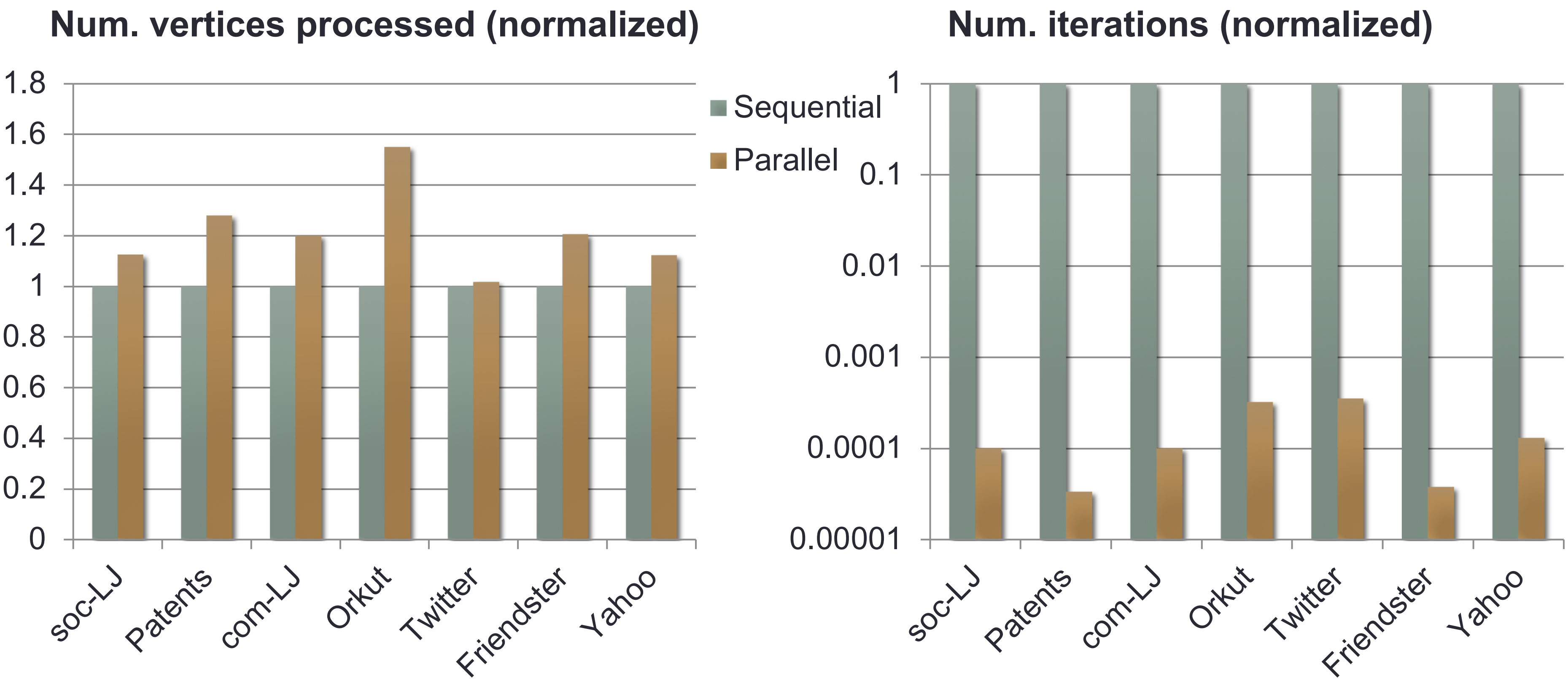
While(any vertex u satisfies **r**[u]/deg(u) ≥ ε)
1) For all vertices u where **r**[u]/deg(u) ≥ ε:
   a) **p**[u] = **p**[u] + α**r**[u]
   b) For all neighbors ngh of u: **r**[ngh] = **r**[ngh] + (1-α)**r**[u]/(2deg(u))
   c) **r**[u] = (1-α)**r**[u]/2

```
sparseSet p = {}, sparseSet r = {}, sparseSet r' = {};        //concurrent hash tables

procedure UpdateNgh(s, d):
    atomicAdd(r'[d], (1-α

procedure UpdateSelf(u):
    p[u] = p[u] + αr[u];

procedure PR-Nibble(G, seed, α, ε):
    r = {(seed, 1.0)};
    while (true):
        active = { u | r[u]/deg(u) ≥ ε};      //vertexSubset
        if active is empty, then break;
        VertexMap(active, UpdateSelf);
        EdgeMap(G, active, UpdateNgh);
        r = r';      //swap roles for next iteration
    return p;
```

Work is only done on "active" vertices and its outgoing edges

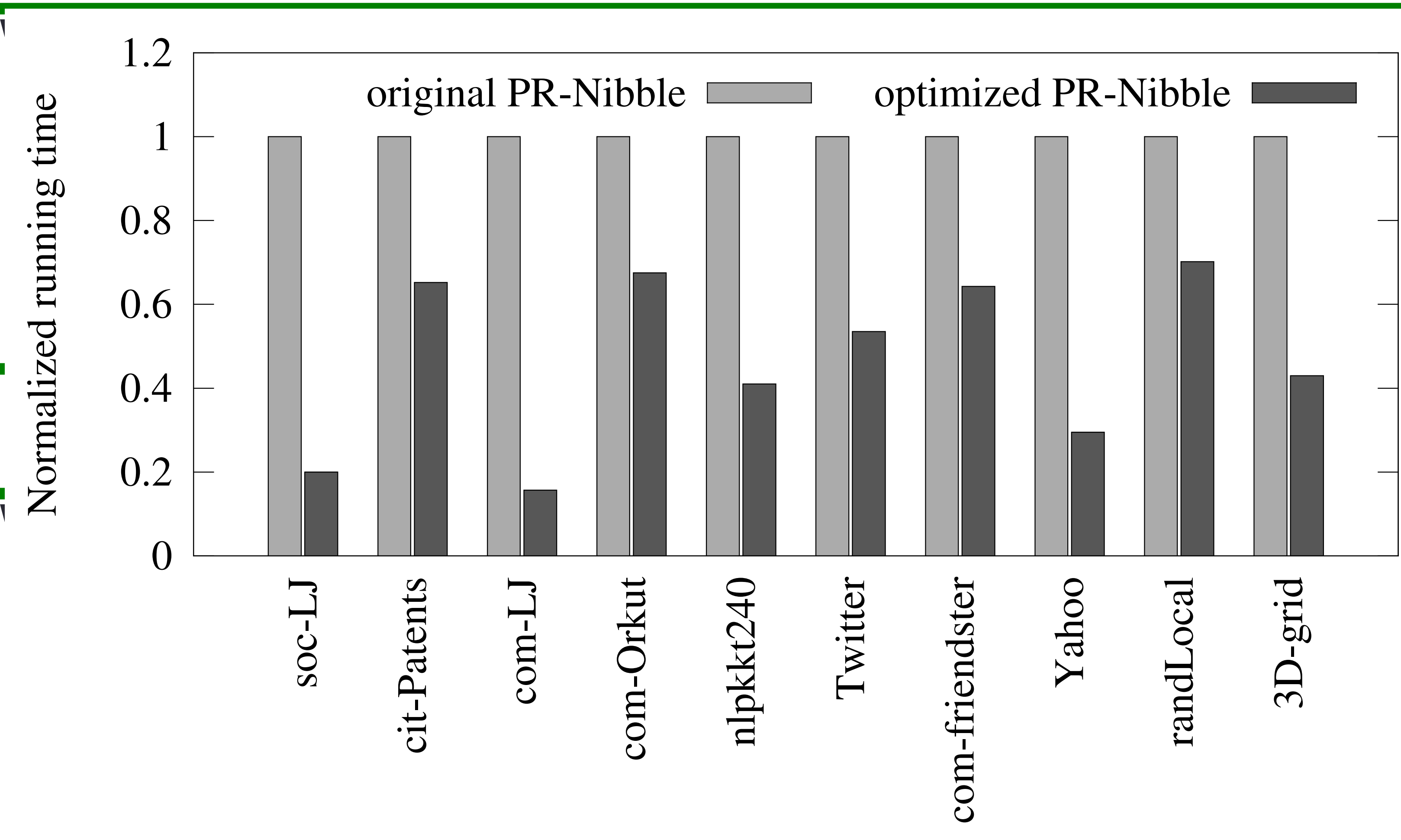
Performance of Parallel PR-Nibble

Parallel PR-Nibble in Practice



- Amount of work slightly higher than sequential
- Number of iterations until termination is much lower!

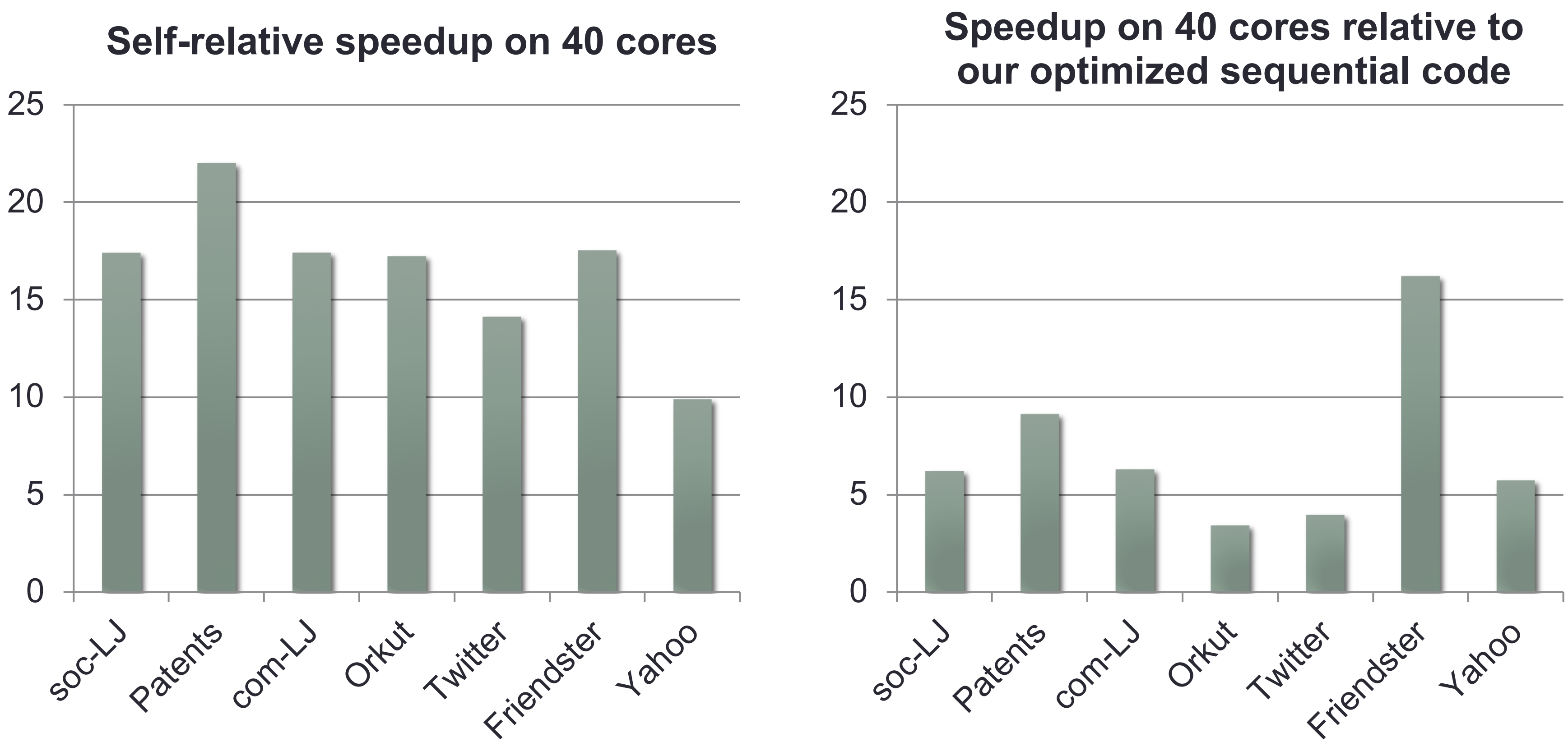
PR-Nibble Optimization



u)

- Gives the same conductance and asymptotic work guarantees as the original algorithm

Parallel PR-Nibble Performance



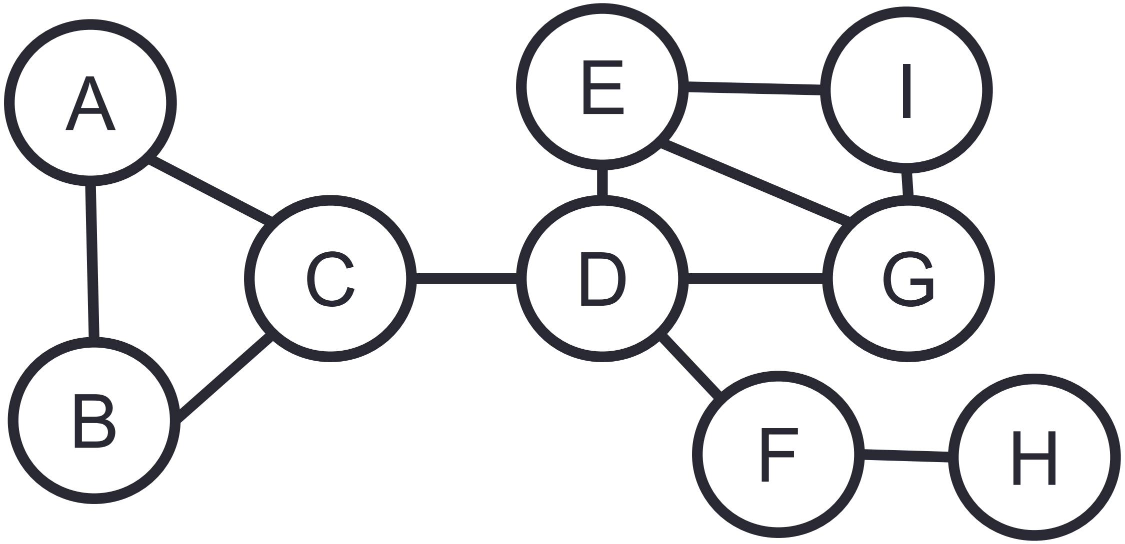
- 10—22x self-relative speedup on 40 cores
- Speedup limited by small active set in some iterations and memory effects
- Running times are in seconds to sub-seconds

Sweep Cut Rounding

Sweep Cut Rounding Procedure

- What to do with the **p** vector?
- Sweep cut rounding procedure:
  - Sort vertices by non-increasing value of **p**[v]/deg(v) (for non-zero entries in **p**)
  - Look at all possible prefixes of sorted order and choose the cut with lowest conductance

Conductance = num. edges leaving cluster / sum of degrees in cluster



Example

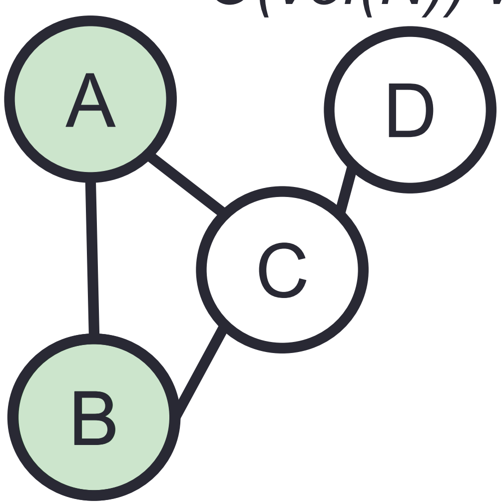
Sorted order = {A, B, C, D}

Cluster	Conductance
{A}	2/2 = 1
{A,B}	2/4 = 0.5
{A,B,C}	1/7 = 0.14
{A,B,C,D}	3/11 = 0.27

Sweep Cut Algorithm

N << # vertices in graph

- Sort vertices by non-increasing value of **p**[v]/deg(v) (for non-zero entries in **p**)
  - *O(N log N) work, where N is number of non-zeros in **p***
- Look at all possible prefixes of sorted order and choose the cut with lowest conductance
  - *Naively takes O(N vol(N)) work, where vol(N) = sum of degrees of non-zero vertices in **p***
  - *O(vol(N)) work algorithm:*



S = {A}
vol = 2
crossingEdges = 2
conductance({A}) = 2/2
S = {A,B}
vol += 2 → vol = 4
crossingEdges--;
crossingEdges++;
conductance({A,B}) = 2/4

```
vol = 0
crossingEdges = 0
S = {}
for each vertex v in sorted order:
    S = S U {v}
    vol += deg(v)
    for each ngh of v:
            if ngh in S: crossingEdges--
            else: crossingEdges++
    conductance(S) = crossingEdges/vol
```

Parallel Sweep Cut

N << # vertices in graph

- Sort vertices by non-increasing value of **p**[v]/deg(v) (for non-zero entries in **p**)
  - *O(N log N) work and O(log N) depth (parallel time), where N is number of non-zeros in **p***
- Look at all possible prefixes of sorted order and choose the cut with lowest conductance
  - *Naively takes O(N vol(N)) work, where vol(N) = sum of degrees of non-zero vertices in **p***
    - *This version is easily parallelizable by considering all cuts independently*
  - *What about parallelizing O(vol(N)) work algorithm?*

O(vol(N)) work parallel algorithm

Conductance = num. edges leaving cluster / sum of degrees in cluster

x — y

If x and y both in prefix, 1 and -1 cancel out in prefix sum
If x in prefix and y is not, only 1 contributes to prefix sum
If neither x nor y in prefix, no contribution

B F — H

Sorted order = {A, B, C, D}

Cluster	Conductance
{A}	2/2 = 1
{A,B}	2/4 = 0.5
{A,B,C}	1/7 = 0.14
{A,B,C,D}	3/11 = 0.27

- Each vertex has rank in sorted order:
  - rank(A) = 1, rank(B) = 2

Hash table insertions: O(N) work and O(log N) depth

- For each incident edge (x, y) of vertex in sorted order, where rank(x) < rank(y), create pairs (1, rank(x)) and (-1, rank(y))

A — B A — C

[(1,1), (-1,2), (1,1), (-1,3), (1,2), (-1,3), (1,3), (-1,4), (1,4), (-1,5), (1,4), (-1,5), (1,4), (-1,5)]

Scan over edges: O(vol(N)) work and O(log vol(N)) depth

- Sort pairs by second value

[(1,1), (1,1), (-1,2), (1,2), (-1,3), (-1,3), (1,

Integer sort: O(vol(N)) work and O(log vol(N)) depth

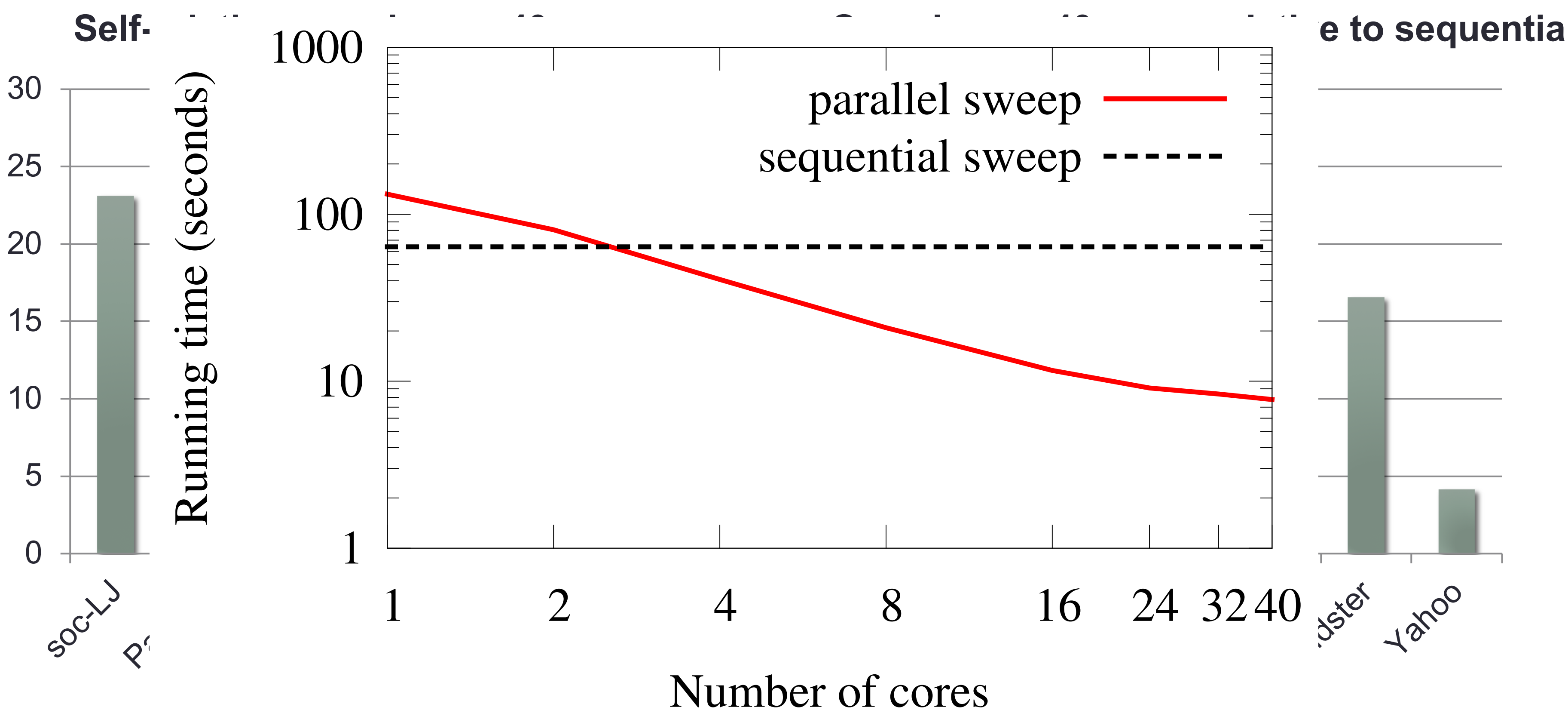
- Prefix sum on first value

[(1,1), (2,1) | (1,2), (2,2) | (1,3), (0,3), (1,3) |

Prefix sum: O(vol(N)) work and O(log vol(N)) depth

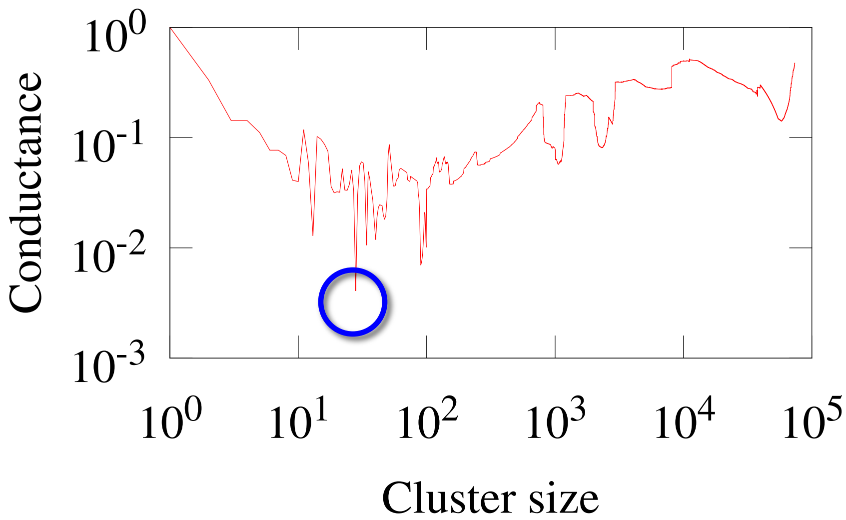
- Get denominator of conductance with prefix sum over degrees

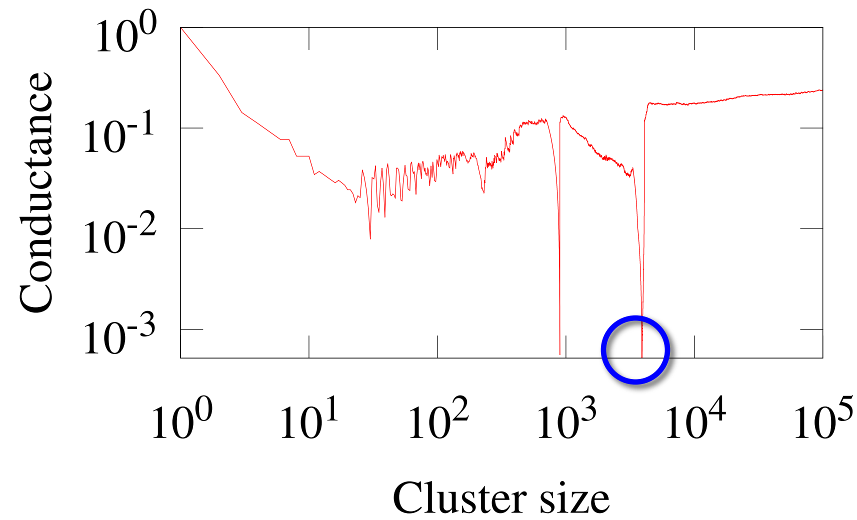
Sweep Cut Performance



- 23—28x speedup on 40 cores
- About a 2x overhead from sequential to parallel
- Outperforms sequential with 4 or more cores

Network Community Profile Plots





42M vertices
1.2B edges

friendster

125M vertices
1.8B edges

- Use parallel algorithms to generate plots for large graphs
- Agrees with conclusions of [Leskovec et al. 2008, 2010, Jeub et al. 2015] that good clusters tend to be relatively small

Summary of our Parallel Algorithms

- Sweep cut
- PageRank-Nibble [Andersen, Chung, and Lang 2006]
- Nibble [Spielman and Teng 2004]
- Deterministic HeatKernel-PageRank [Kloster and Gleich 2014]
- Randomized HeatKernel-PageRank [Chung and Simpson 2015]

- Based on iteratively processing sets of “active” vertices in parallel
- Use concurrent hash tables and Ligra’s functionality to get local running times
- We prove theoretically that parallel work asymptotically matches sequential work, and obtain low depth (parallel time) complexity