Architecture- and Workload-Aware Graph (Re)Partitioning

Thesis Defense
Angen Zheng

Thesis Committee

Alexandros Labrinidis
CS, Pitt

Panos K. Chrysanthis
CS, Pitt

Jack Lange
CS, Pitt

Peyman Givi
SSOE, Pitt

Patrick Pisciuneri
SSOE, Pitt
Big Graphs

Web Network

Social Network

DNA Interaction Network

Follow Network

User-Product Network
Distributed Computation

Giraph++

GraphLab

Pregel

GoFFish

Apache Hama
A Balanced Partitioning = Even Load Distribution
Minimal Edge-Cut = Minimal Data Comm
Thesis Statement

Architecture- and workload-aware graph partitioning enables efficient distributed graph computation in modern HPC clusters.
## Research Overview

### Architecture- and Workload-Aware Graph (Re)Partitioning

<table>
<thead>
<tr>
<th>Method</th>
<th>Conference/Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aragon</td>
<td>BigGraphs’14</td>
<td>small dynamic graphs</td>
</tr>
<tr>
<td>Paragon</td>
<td>EDBT’16</td>
<td>median-size dynamic graphs</td>
</tr>
<tr>
<td>Planar</td>
<td>ICDE’16</td>
<td>large dynamic graphs</td>
</tr>
<tr>
<td>Planar+</td>
<td>To submit’17</td>
<td>large dynamic graphs</td>
</tr>
<tr>
<td>Argo</td>
<td>BigData’16</td>
<td>static graphs</td>
</tr>
<tr>
<td>Sargon</td>
<td>ICDE’17</td>
<td>skew-resistant</td>
</tr>
</tbody>
</table>
Planar+: Parallel Lightweight Architecture-Aware Graph Repartitioning

Angen Zheng · Patrick Pisciuneri · Alexandros Labrinidis · Panos K. Chrysanthis · Jack Lange · Peyman Givi

<table>
<thead>
<tr>
<th>Architecture- and Workload-Aware Graph (Re)Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aragon</strong> [BigGraphs’14] (small dynamic graphs)</td>
</tr>
<tr>
<td><strong>Planar+</strong> [To submit’17] (large dynamic graphs)</td>
</tr>
<tr>
<td><strong>Paragon</strong> [EDBT’16] (median-size dynamic graphs)</td>
</tr>
<tr>
<td><strong>Argo</strong> [BigData’16] (static graphs)</td>
</tr>
<tr>
<td><strong>Planar</strong> [ICDE’16] (large dynamic graphs)</td>
</tr>
<tr>
<td><strong>Sargon</strong> [ICDE’17] (skew-resistant)</td>
</tr>
</tbody>
</table>
Planar+: Overview

❖ Optimizations

➢ Memorization technique
➢ Eliminating per step physical vertex migration
➢ Optimized network comm cost measurement
➢ Optimized vertex gain computation
Planar: Overview

Phase-1: Logical Vertex Migration
- Phase-1a: Minimizing Comm Cost
- Phase-1b: Ensuring Balanced Partitions

Phase-2: Physical Vertex Migration

Phase-3: Convergence Check

Migration Planning
- What vertices to move?
- Where to move?

Perform the Migration Plan

Still beneficial?
Planar+: Eliminating Per Step Physical Vertex Migration

- **Phase-1: Logical Vertex Migration**
  - Phase-1a: Minimizing Comm Cost
  - Phase-1b: Ensuring Balanced Partitions

- **Phase-2: Vertex Location Update**
  - Each vertex has up-to-date locations of their neighbors

- **Phase-3: Convergence Check**
  - Still beneficial?

**Migration Planning**
- What vertices to move?
- Where to move?
Planar+: Eliminating Per Step Physical Vertex Migration

Starts Repartitioning

\( S_k \) \( S_{k+1} \) \( S_{k+2} \) \( S_{k+4} \) \( S_{k+5} \)

Converge

Physical Vertex Migration
Planar+: Optimized Network Comm Cost Measurement

- five 20-core machines
- one process per core
- two CPU sockets per machine

<table>
<thead>
<tr>
<th></th>
<th>Naive Solution (Measuring all pair-wise cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Pairs Measured</td>
<td>(100 * 99) / 2 = 4950</td>
</tr>
</tbody>
</table>
Planar+: Optimized Network Comm Cost Measurement

Intra-Socket

Inter-Machine

Inter-Socket

Machine 0

Machine 1

Socket 0

Socket 1

Socket 0

Socket 1

core...core
L1...L1
L2...L2

core...core
L1...L1
L2...L2

core...core
L1...L1
L2...L2

L3

L3

L3

QPI/HT

QPI/HT

QPI/HT

Memory Controller

Memory Controller

Memory Controller

Memory Controller

Memory

Memory

Memory

Memory
Planar+: Optimized Network Comm Cost Measurement

- five 20-core machines
- one process per core
- two CPU sockets per machine

<table>
<thead>
<tr>
<th># of Pairs Measured</th>
<th>Naive Solution (Measuring all pair-wise cost)</th>
<th>Optimized Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(100 \times 99) / 2 = 4950$</td>
<td>25</td>
</tr>
</tbody>
</table>

- 10 pairs inter-machine
- 5 pairs inter-socket
- 10 pairs intra-socket
Planar+: Optimized Vertex Gain Computation

\[ g^{i,j}(v) = \alpha \ast (\text{comm}(v, P_i) - \text{comm}(v, P_j)) - \text{mig}(v, P_i, P_j) \]

Comm cost v would cause if it is assigned to partition i

Cost of migrating v from partition i to partition j

\[ \text{comm}(v, P_i) = \sum_{k=1 \text{ and } k \neq i}^{n} d_{ext}(v, P_k) \ast c(P_i, P_k) \]

Amount of data v communicates with partition k

Network comm cost between partition i and partition k

O(n), where n is # of partitions.
Planar+: Optimized Vertex Gain Computation

\[
\text{comm}(v, P_i) = \sum_{k=1 \text{ and } k \neq i}^{n} d_{ext}(v, P_k) \times c(P_i, P_k)
\]

Amount of data \(v\) communicates with \text{partition k}

Network comm cost between \text{partition i and partition k}

Optimized Solution:

\[
\sum_{k=1 \text{ and } k \neq i}^{m} d_{mach}(v, M_k) \times c_{mach}(M_i, M_k)
\]

Amount of data \(v\) communicates with \text{machine k}

Network comm cost between \text{machine i and machine k}

\(O(n) \rightarrow O(m), \text{ where m is # of machines used.}\)
Planar+: Optimized Vertex Gain Computation

\[
\sum_{k=1 \text{ and } k \neq i}^{m} d_{mach}(v, M_k) \ast c_{mach}(M_i, M_k)
\]
Planar+: PageRank Execution Time ($\lambda=1$)

- **PageRank**: 20 iterations
- **Friendster**: $|V|=124M$, $|E|=3.68$
- **120 Partitions**: six 20-core machines
- **Initial Partitioner**: LDG

---

Hours CPU Time Saving

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PARAGON</td>
<td>25h</td>
</tr>
<tr>
<td>PLANAR</td>
<td>27h</td>
</tr>
<tr>
<td>PLANAR+</td>
<td>43h</td>
</tr>
<tr>
<td>uniPLANAR+</td>
<td>10h</td>
</tr>
</tbody>
</table>

✓ Architecture-aware solutions performed better.
Planar+: PageRank Execution Time ($\lambda=1$)

- PageRank: 20 iterations
- Friendster: $|V|=124M$, $|E|=3.6B$
- 120 Partitions: six 20-core machines
- Initial Partitioner: LDG

**Hours CPU Time Saving**

- PARAGON: 25h
- PLANAR: 27h
- PLANAR+: 43h
- uniPLANAR+: 10h

Planar+ was
- 6.5x faster than Planar
- 3.5x faster than Paragon
Planar+: Summary

★ Planar+

- Lightweight Architecture-Aware Graph Repartitioner
  - Communication Heterogeneity
  - Shared Resource Contention
- Achieved up to hours CPU time saving.
- Scaled well in terms of
  - Graph size (up to two billion-edge graphs).
  - Number of partitions (up to 240 partitions).
## ARGO: Architecture-Aware Graph Partitioning

Angen Zheng, Alexandros Labrinidis, Panos K. Chrysanthis, Jack Lange  
Department of Computer Science, University of Pittsburgh  
{anz28, labrinid, panos, jacklange}@cs.pitt.edu

*IEEE International Conference on Big Data (IEEE BigData), 2016*

<table>
<thead>
<tr>
<th>Architecture- and Workload-Aware Graph (Re)Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aragon</strong> [BigGraphs’14] (small dynamic graphs)</td>
</tr>
<tr>
<td><strong>Planar+</strong> [To submit’17] (large dynamic graphs)</td>
</tr>
<tr>
<td><strong>Paragon</strong> [EDBT’16] (median-size dynamic graphs)</td>
</tr>
<tr>
<td><strong>Argo</strong> [BigData’16] (static graphs)</td>
</tr>
<tr>
<td><strong>Planar</strong> [ICDE’16] (large dynamic graphs)</td>
</tr>
<tr>
<td><strong>Sargon</strong> [ICDE’17] (skew-resistant)</td>
</tr>
</tbody>
</table>
Argo: Motivation

- Highly nonuniform network comm costs
- Multicore clusters with high-speed networks
  - Contention on the memory subsystems
    - Intra-node data comm increases contention
    - Inter-node data comm alleviates contention
Given $G=(V, E)$

$$P = \{ P_i : \bigcup_{i=1}^{n} P_i = V \text{ and } P_i \cap P_j = \emptyset \text{ for any } i \neq j \}$$

Balancing Load:

$$w(P_i) \approx \frac{1}{n} \sum_{v \in V} w(v)$$

Minimizing Communication:

$$\sum_{e=(u,v) \in E \land u \in P_i \land v \in P_j \land i \neq j} w(e) \ast c[i][j]$$

Vertex Weight

Edge Weight

Network comm cost or Degree of contention
Argo: **Graph Partitioning Model**

Streaming Graph Partitioning Model (Stanton et. al, KDD’12)
Argo: Incorporating Heterogeneity

Place vertex, $v$, to a partition, $Pi$, that maximize:

$$\frac{1}{\text{comm}(v, P_i) + 1} \times \left(1 - \frac{w(P_i)}{C(P_i)}\right)$$

- **Weighted Edge-Cut**
- **Penalize the placement based on the load of Pi**

 ✓ Weighted by the relative network comm cost, Argo will avoid
  - edge-cut across nodes
  - inter-node data comm
Argo: Incorporating Contentiousness

Place vertex, $v$, to a partition, $Pi$, that maximize:

$$\frac{1}{\text{comm}(v, P_i) + 1} \times \left(1 - \frac{w(P_i)}{C(P_i)}\right)$$

- **Weighted Edge-Cut**
- **Penalize the placement based on the load of Pi**

✓ Weighted by **the degree of contention**, Argo will avoid
  - edge-cut across cores of the same node
  - excess intra-node data comm
Argo: Incorporating Contentiousness

Degree of Contention
($\lambda \in [0, 1]$)

$c(P_i, P_j) = c(P_i, P_j) + \lambda \times s$

<table>
<thead>
<tr>
<th>Bottleneck</th>
<th>Network</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda=0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda=1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Refined Intra-Node Network Comm Cost

Original Intra-Node Network Comm Cost

Maximal Inter-Node Network Comm Cost

✓ Weighted by the refined network comm cost, Argo will avoid
  ○ edge-cut across cores of the same node
  ○ excess intra-node data comm
Argo: **SSSP Exec. Time** on Orkut dataset

- **SSSP:** 10 source vertices
- **Orkut:** \(|V| = 3M, |E| = 234M\)
- **60 Partitions:** three 20-core machines

![Bar chart showing normalized execution time for different methods: METIS, LDG, ARGO-N, ARGO-M.](chart)

✔️ Contention on the memory subsystems was the bottleneck.
- **ARGO-M** had the **lowest** SSSP execution time.
- **ARGO-N** had the **longest** SSSP execution time.
Argo: SSSP LLC Misses on Orkut dataset

- SSSP: 10 source vertices
- Orkut: |V| = 3M, |E| = 234M
- 60 Partitions: three 20-core machines

![Graph showing normalized LLC misses for METIS, LDG, ARGO-N, and ARGO-M with 4x, 3x, 6x, 9x, 12x, 1x, 9x, 1.2x, 1x percentages at different partition sizes.]

- ARGO-M had the lowest LLC Misses.
- ARGO-N had the highest LLC Misses.
Argo: SSSP Comm Vol. on Orkut dataset

- SSSP: 10 source vertices
- Orkut: $|V| = 3M$, $|E| = 234M$
- 60 Partitions: three 20-core machines

<table>
<thead>
<tr>
<th></th>
<th>Intra-Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>METIS</td>
<td>69%</td>
</tr>
<tr>
<td>LDG</td>
<td>49%</td>
</tr>
<tr>
<td>ARGO-N</td>
<td>70%</td>
</tr>
</tbody>
</table>

✓ ARGO-M had the lowest intra-node communication volume.
✓ ARGO-N had the highest inter-node communication volume.
Understanding the Performance Impact of Memory Subsystem Contention

<table>
<thead>
<tr>
<th></th>
<th>METIS and LDG</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph Partitioners</strong></td>
<td>BFS, SSSP, and PageRank</td>
</tr>
<tr>
<td><strong>Graph Workloads</strong></td>
<td>Orkut ($</td>
</tr>
<tr>
<td><strong>Graph Dataset</strong></td>
<td>16 (one partition per core)</td>
</tr>
</tbody>
</table>
Understanding the Performance Impact of Memory Subsystem Contention

<table>
<thead>
<tr>
<th>m:s:c</th>
<th>SSSP Execution Time (s)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>METIS</td>
<td>LDG</td>
<td></td>
</tr>
<tr>
<td>1:2:8</td>
<td>633</td>
<td>2,632</td>
<td></td>
</tr>
<tr>
<td>2:2:4</td>
<td>654</td>
<td>2,565</td>
<td></td>
</tr>
<tr>
<td>4:2:2</td>
<td>521</td>
<td>631</td>
<td></td>
</tr>
<tr>
<td>8:2:1</td>
<td>222</td>
<td>280</td>
<td></td>
</tr>
</tbody>
</table>

- **m**: # of machines used
- **s**: # of sockets used per machine
- **c**: # of cores used per socket

- ✓ Denser configurations had longer execution time.
  - Contention on the memory subsystems impacted performance.
  - Network may not always be the bottleneck.
Understanding the Performance Impact of Memory Subsystem Contention

<table>
<thead>
<tr>
<th>m:s:c</th>
<th>SSSP Execution Time (s)</th>
<th></th>
<th>SSSP LLC Misses (in Millions)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>METIS</td>
<td>LDG</td>
<td>METIS</td>
<td>LDG</td>
</tr>
<tr>
<td>1:2:8</td>
<td>633</td>
<td>2,632</td>
<td>10,292</td>
<td>44,117</td>
</tr>
<tr>
<td>2:2:4</td>
<td>654</td>
<td>2,565</td>
<td>10,626</td>
<td>44,689</td>
</tr>
<tr>
<td>4:2:2</td>
<td>521</td>
<td>631</td>
<td>2,541</td>
<td>1,061</td>
</tr>
<tr>
<td>8:2:1</td>
<td>222</td>
<td>280</td>
<td>96</td>
<td>187</td>
</tr>
</tbody>
</table>

✔ Denser configurations had longer execution time.
  ○ Contention on the memory subsystems impacted performance.
  ○ Network may not always be the bottleneck.
Understanding the Performance Impact of Memory Subsystem Contention

<table>
<thead>
<tr>
<th>m:s:c</th>
<th>SSSP Execution Time (s)</th>
<th></th>
<th>SSSP LLC Misses (in Millions)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>METIS</td>
<td>LDG</td>
<td>METIS</td>
<td>LDG</td>
</tr>
<tr>
<td>1:2:8</td>
<td>633</td>
<td>2,632</td>
<td>10,292</td>
<td>44,117</td>
</tr>
<tr>
<td>2:2:4</td>
<td>654</td>
<td>2,565</td>
<td>10,626</td>
<td>44,689</td>
</tr>
<tr>
<td>4:2:2</td>
<td>521</td>
<td>631</td>
<td>2,541</td>
<td>1,061</td>
</tr>
<tr>
<td>8:2:1</td>
<td>222</td>
<td>280</td>
<td>96</td>
<td>187</td>
</tr>
</tbody>
</table>

✅ Denser configurations had longer execution time.
   ○ Contention on the memory subsystems impacted performance.
   ○ Network may not always be the bottleneck.
Understanding the Performance Impact of Memory Subsystem Contention

<table>
<thead>
<tr>
<th>m:s:c</th>
<th>SSSP Execution Time (s)</th>
<th>M</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:2:8</td>
<td>633</td>
<td>2,632</td>
<td></td>
</tr>
<tr>
<td>2:2:4</td>
<td>654</td>
<td>2,565</td>
<td></td>
</tr>
<tr>
<td>4:2:2</td>
<td>521</td>
<td>631</td>
<td></td>
</tr>
<tr>
<td>8:2:1</td>
<td>222</td>
<td>280</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m:s:c</th>
<th>SSSP LLC Misses (in Millions)</th>
<th>M</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:2:8</td>
<td>10,292</td>
<td>44,117</td>
<td></td>
</tr>
<tr>
<td>2:2:4</td>
<td>10,626</td>
<td>44,689</td>
<td></td>
</tr>
<tr>
<td>4:2:2</td>
<td>2,541</td>
<td>1,061</td>
<td></td>
</tr>
<tr>
<td>8:2:1</td>
<td>96</td>
<td>187</td>
<td></td>
</tr>
</tbody>
</table>

✓ METIS had lower execution time and LLC misses than LDG.
   ○ Edge-cut matters.
   ○ Higher edge-cut-->higher comm-->higher contention
Argo: **Summary**

✓ **Findings**
  - Network is not always the bottleneck.
  - Contention on the memory subsystems may impact the performance a lot.
    - due to excess intra-node data communication

✓ **ARGO**
  - Avoids contention by offloading excess intra-node data comm across nodes.
  - Achieved up to **11x** improvement on real-world workloads.
  - Scaled well in terms of
    - Graph size (up to **2 billion-edge** graphs)
    - # of partitions (up to **200** partitions)
Skew-Resistant Graph Partitioning

Aragon [BigGraphs’14] (small dynamic graphs)
Paragon [EDBT’16] (median-size dynamic graphs)
Planar [ICDE’16] (large dynamic graphs)

Planar+ [To submit’17] (large dynamic graphs)
Argo [BigData’16] (static graphs)
Sargon [ICDE’17] (skew-resistant)

Angen Zheng
University of Pittsburgh
anz28@cs.pitt.edu

Alexandros Labrinidis
University of Pittsburgh
labrinid@cs.pitt.edu

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

33rd IEEE International Conference on Data Engineering (ICDE), 2017
A Balanced Partitioning = Even Load Distribution
Minimal Edge-Cut = Minimal Data Comm

Assumption: Vertices of the graph are always active.
A Balanced Partitioning = Even Load Distribution
Minimal Edge-Cut = Minimal Data Comm

Assumption broken by:
- Traversal-Style Graph Workloads
- Multiphase Graph Workloads
- Graph Workloads with Seasonal Access Pattern
- Graphs with Varying Hotness on the Vertices

Assumption: Vertices of the graph are always active.
### Traversal-Style Graph Workloads: Active Vertex Distribution Across Supersteps

<table>
<thead>
<tr>
<th>Workloads</th>
<th>BFS &amp; SSSP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>com-orkut</strong></td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>V</td>
</tr>
<tr>
<td>(</td>
<td>E</td>
</tr>
<tr>
<td>Max Degree</td>
<td>33,313</td>
</tr>
<tr>
<td>Avg. Degree</td>
<td>76.281</td>
</tr>
<tr>
<td>Max Diameter</td>
<td>10</td>
</tr>
<tr>
<td>Avg. Diameter</td>
<td>5.4489</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supersteps</th>
<th>BFS</th>
<th>SSSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>72</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>5,871</td>
<td>4,663</td>
</tr>
<tr>
<td>3</td>
<td>215,425</td>
<td>297,943</td>
</tr>
<tr>
<td>4</td>
<td>1,753,891</td>
<td>1,421,993</td>
</tr>
<tr>
<td>5</td>
<td>1,088,870</td>
<td>1,229,917</td>
</tr>
<tr>
<td>6</td>
<td>8,242</td>
<td>117,496</td>
</tr>
<tr>
<td>7</td>
<td>69</td>
<td>383</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Not all vertices are always active.
Highly skewed active vertex distribution across supersteps.
Traversal-Style Graph Workloads:
Active Vertex Distribution Across Partitions

Highly skewed active vertex distribution across partitions

Distribute vertices active in the same time step evenly across partitions.
Traversal-Style Graph Workloads:
High-Degree Vertex Distribution Across Partitions

Highly Skewed high-degree vertex distribution across partitions.

Distribute high-degree vertices evenly across partitions.
Sargon: Skew-Resistant Graph Partitioning

Given $G=(V, E)$

- Assign a label vector to each vertex to indicate:
  - the time periods the vertex is active in
  - whether it is a high- or low-degree vertex
  - the hotness of the vertex

$$P = \{P_i : \bigcup_{i=1}^{n} P_i = V \text{ and } P_i \cap P_j = \emptyset \text{ for any } i \neq j\}$$

Balancing Load:

$$w(P_i, l) \approx \frac{1}{n} \sum_{v^l \in V} w(v^l), \text{ for any label } l$$

Minimizing Communication:

$$\sum_{e=(u,v) \in E \land u \in P_i \land v \in P_j \land i \neq j} w(e)$$
Sargon: Graph Partitioning Model

Streaming Graph Partitioning Model (Stanton et. al, KDD'12)
Sargon: Vertex Placement Heuristic

Place vertex, $v$, to a partition, $P_i$, that maximize:

$$
\sum_{e=(u,v) \in E \land u \in P_i} w(e) \times \text{loadFactor}(P_i, v)
$$

$$
\sum_{l \in \{v's \ labels\}} (1 - \frac{w(P_i, l)}{C(P_i, l)})
$$

Sargon will try its best to:
- Distribute vertices of the same labels evenly across partitions.
- Minimize the edgecut.

Extract the labels from the execution trace of the workload.
Sargon: Assumption Validation

| Workloads            | BFS and SSSP (one randomly selected source vertex) | Dataset                | Orkut (|V|=3M, |E|=234M) |
|----------------------|------------------------------------------------|------------------------|------------------|
| # of Traces Collected| 5                                               |                        |                  |
| Similarity           | Percentage of the vertices overlapped in the peak superstep |                        |                  |

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Avg. Similarity</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>60.80%</td>
<td>8.43%</td>
</tr>
<tr>
<td>SSSP</td>
<td>64.73%</td>
<td>10.63%</td>
</tr>
</tbody>
</table>

Workload execution traces are quite similar. Quite predictable runtime characteristics.
Sargon: SSSP Execution Time

- SSSP: 100 source vertices
- Orkut: |V|=3M, |E|=234M
- 60 Partitions: three 20-core machines

✓ Up to 2x speedups (hours CPU time saving).
Sargon: Summary

✓ A study of the traversal-style graph workloads
  ○ Not all vertices are always active.
  ○ The distribution of active vertices across supersteps may be highly skewed.
  ○ The execution traces of the workloads are quite similar in many cases.

✓ SARGON
  ○ Avoids time-varying skewness of the target workload.
  ○ Achieved up to $2\times$ speedups on real-world workloads.
Conclusions

Architecture- and Workload-Aware Graph (Re)Partitioning

- **Aragon** [BigGraphs’14] (small dynamic graphs)
- **Paragon** [EDBT’16] (median-size dynamic graphs)
- **Planar** [ICDE’16] (large dynamic graphs)
- **Planar+** [To submit’17] (large dynamic graphs)
- **Argo** [BigData’16] (static graphs)
- **Sargon** [ICDE’17] (skew-resistant)

(P)aragon: [https://github.com/admtlab/paragon](https://github.com/admtlab/paragon)
Planar+: [https://github.com/admtlab/planar](https://github.com/admtlab/planar)

Thanks!

**Acknowledgments:**
- Thesis Committee
- Christos Faloutsos

**Funding:**
- NSF OIA-1028162
- NSF CBET-1250171
- NSF CBET-1609120
Publications

[1]. Planar+: Parallel Lightweight Architecture-Aware Graph Repartitioning,
Angen Zheng, Patrick Pisciuneri, Alexandros Labrinidis, Panos K. Chrysanthis, Jack Lange, Peyman Givi
To submit, 2017

[2]. Skew-Resistant Graph Partitioning (Short Paper)
Angen Zheng, Alexandros Labrinidis, Christos Faloutsos
33nd IEEE International Conference on Data Engineering (ICDE), 2017

[3]. Argo: Architecture-Aware Graph Partitioning
Angen Zheng, Alexandros Labrinidis, Panos K. Chrysanthis, Jack Lange
IEEE International Conference on Big Data (IEEE BigData), 2016

[4]. Planar: Parallel Lightweight Architecture-Aware Adaptive Graph Repartitioning
Angen Zheng, Alexandros Labrinidis, Panos K. Chrysanthis
32nd IEEE International Conference on Data Engineering (ICDE), 2016

[5]. Paragon: Parallel Architecture-Aware Graph Partition Refinement Algorithm
Angen Zheng, Alexandros Labrinidis, Patrick Pisciuneri, Panos K. Chrysanthis, Peyman Givi
19th International Conference on Extending Database Technology (EDBT), 2016

[6]. Architecture-Aware Graph Repartitioning for Data-Intensive Scientific Computing
Angen Zheng, Alexandros Labrinidis, Panos K. Chrysanthis
1st International Workshop on High Performance Big Graph Data Management, Analysis, and Mining, 2014