Theoretically and Practically Efficient Nucleus Decomposition

Jessica Shi
(MIT / Google)

Laxman Dhulipala
(University of Maryland)

Julian Shun
(MIT)
How do we cluster a graph?

▷ A fundamental idea:

How well-connected are certain nodes or subsets of nodes in a graph?
“Well-connected” nodes

- **k-core**: Repeatedly find + “delete” min degree vertex

Formally: A k-core is an induced subgraph where every vertex has degree at least k
A problem with k-core

- **k-core**: Repeatedly find + “delete” min degree vertex

Entire graph is in a 3-core
s-clique peeling

- s-clique degree: Number of s-cliques each vertex participates in
- s-clique peeling: Repeatedly find + “delete” min s-clique degree vertex

3-triangle cores (s = 3)
(r, s)-nucleus decomposition

- **s-clique degree of a r-clique**: Number of s-cliques each r-clique participates in
- **(r, s)-nucleus decomposition**: Repeatedly find + “delete” r-clique with min s-clique degree

2-(( , ) nuclei
(r = 2, s = 3)

(r = 2, s = 3 is also known as k-truss)
\((r, s)\)-nucleus decomposition

\((1, 2)\)-nuclei = k-core

\((1, 3)\)-nuclei = triangle-peeling

facebook graph (88k edges)

Sariyuce, Seshadhri, Pinar, Catalyurek (2017)
(r, s)-nucleus decomposition

(3, 4)-nuclei

facebook graph (88k edges)

Sariyuce, Seshadhri, Pinar, Catalyurek (2017)
Main results

▷ New shared-memory parallel algorithms for \textit{nucleus decomposition} with \textit{strong theoretical guarantees}

▷ Comprehensive evaluation, showing we \textit{outperform state-of-the-art parallel algorithms} by a couple orders of magnitude
Computational barriers: Sequential subgraph decomposition can be slow

- **Environment**: 30-core GCP instance (2-way hyperthreading), 240 GiB main memory

<table>
<thead>
<tr>
<th>Graph</th>
<th># Edges</th>
<th>Sequential (3, 4)-nucleus decomp [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>as-skitter</td>
<td>11 million</td>
<td>8.5 minutes</td>
</tr>
<tr>
<td>livejournal</td>
<td>34 million</td>
<td>3.3 hours</td>
</tr>
<tr>
<td>orkut</td>
<td>117 million</td>
<td>&gt; 6 hours</td>
</tr>
</tbody>
</table>

- **Goal**: < 15 min

Theoretically efficient algorithms are fast

- Previous parallel nucleus decomposition [2]: Not theoretically efficient

![Bar chart showing running times for different networks and methods.]

- as-skitter (11M):
  - Sequential [1]: 8.4 min
  - Parallel [2]: 3.7 min
  - Our parallel (theoretically efficient): 50 sec

- livejournal (34M):
  - Parallel [2]: 3.2 hrs
  - Our parallel (theoretically efficient): 14 min

- orkut (117M):
  - Timed out
  - Our parallel (theoretically efficient): 21 min

---

Practical optimizations

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>as-skitter (11M)</td>
<td>8.4 min</td>
<td>3.7 min</td>
<td>50 sec</td>
<td>24 sec</td>
</tr>
<tr>
<td>livejournal (34M)</td>
<td>3.3 hrs</td>
<td>3.2 hrs</td>
<td>14 min</td>
<td>3.3 min</td>
</tr>
<tr>
<td>orkut (117M)</td>
<td>Timed out</td>
<td>1.3 hrs</td>
<td>21 min</td>
<td>13 min</td>
</tr>
</tbody>
</table>

Preliminaries
Preliminaries

- **Work** = total # operations
- **Span** = longest dependency path
- **Running time** $\leq \frac{\text{work}}{\# \text{processors}} + O(\text{span})$
- **Work-efficient** = work matches best sequential time complexity
Graph orientation

- $\alpha = \text{arboricity} = \text{minimum } \# \text{ of spanning forests needed to cover all edges of the graph}$
  - Upper bounded by $O(\sqrt{m})$ where $m = \# \text{ edges}$

- c-orientation: Direct graph such that each vertex’s out-degree is upper bounded by $c$

- Arboricity orientation: $O(\alpha)$-orientation

- Our prior work: Two theoretically efficient arboricity orientation algorithms $[1]$

Parallel nucleus decomposition
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques

No 4-cliques: cdg
One 4-clique: All triples in \{a,b,e,f\} except abe
Two 4-cliques: All triples in \{a,b,c,d,e\} except abe
Three 4-cliques: abe
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques
(r, s)-nucleus decomposition (r=3, s=4)

Direct the graph (DG) using an arboricity orientation
Count # s-cliques per r-clique using DG
Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
While not all r-cliques have been peeled:
  ○ Peel set of r-cliques with minimum s-clique count
  ○ Update s-clique counts of remaining r-cliques

No 4-cliques:
One 4-clique: All triples in \{a,b,e,f\} except abe
Two 4-cliques: All triples in \{a,b,c,d,e\} except abe
Three 4-cliques: abe
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques

No 4-cliques:
One 4-clique: All triples in \{a,b,e,f\} except abe
Two 4-cliques: All triples in \{a,b,c,d,e\} except abe
Three 4-cliques: abe
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques

No 4-cliques:
One 4-clique:
Two 4-cliques: All triples in \{a,b,c,d,e\}
Three 4-cliques:
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques

No 4-cliques:
One 4-clique:
Two 4-cliques: All triples in \{a,b,c,d,e\}
Three 4-cliques:

 ![Diagram of a graph with vertices a, b, c, d, e]
(r, s)-nucleus decomposition (r=3, s=4)

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques

No 4-cliques:
One 4-clique:
Two 4-cliques:
Three 4-cliques:
(r, s)-nucleus decomposition

0(m) work, O(log^2 n) span

0(mα^{s-2}) work, O(s log n) span whp

▷ Direct the graph (DG) using an arboricity orientation
▷ Count # s-cliques per r-clique using DG
▷ Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
▷ While not all r-cliques have been peeled:
  ○ Peel set of r-cliques with minimum s-clique count
  ○ Update s-clique counts of remaining r-cliques

Shi, Dhulipala, Shun (2021)
(r, s)-nucleus decomposition

$O(m)$ work, $O(\log^2 n)$ span

$O(m\alpha^{s-2})$ work, $O(s \log n)$ span whp

- Direct the graph (DG) using an arboricity orientation
- Count # s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques

Subgoal 1

Subgoal 2
How do we peel r-cliques?

- **Subgoal 1**: A way to keep track of r-cliques with min s-clique count

- **In theory**: Use a batch-parallel Fibonacci heap \[1\]
  -\( k \) insertions: \( O(k) \) amortized expected work, \( O(\log n) \) span whp
  - Extract min: \( O(\log n) \) amortized expected work, \( O(\log n) \) span whp

- **In practice**: Fibonacci heaps are not efficient
  - **Julienne**: Efficient parallel bucketing structure \[2\]

---

(r, s)-nucleus decomposition

\[ O(m) \text{ work, } O(\log^2 n) \text{ span} \]

\[ O(m\alpha^{s-2}) \text{ work, } O(s \log n) \text{ span whp} \]

\[ O(m\alpha^{r-2} + \rho \log n) \text{ amortized expected work, } O(\rho \log n) \text{ span whp} \]

where \( \rho = \# \text{ rounds to peel entire graph} \)

Subgoal 2

- Direct the graph (DG) using an arboricity orientation
- Count \# s-cliques per r-clique using DG
- Construct a bucketing structure mapping r-cliques to a bucket based on \# s-cliques
- While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques
How do we update s-clique counts?

- **Subgoal 2:** A way to update s-clique counts after “deleting” r-cliques
  - **In theory and practice:** We use a key lemma that improves upon the previous best theoretical bounds for sequential nucleus decomposition
  - **In practice:** Also use software optimizations
Theoretically: Update s-clique counts

- **Subgoal 2**: A way to update s-clique counts after “deleting” r-cliques

- Modify parallel s-clique counting subroutine to efficiently obtain updated s-clique counts from “deleted” r-cliques

- **Theorem**: Over all c-cliques in a graph $C_c = \{v_1, \ldots, v_c\}$,
  \[ \sum_{C_c} \min_{1 \leq i \leq c} \deg(v_i) = O(m\alpha^{c-1}). \] [1]

[1] Eden, Ron, Seshadhri 2020
Theoretically: Update s-clique counts

**Theorem:** Over all c-cliques in a graph $C_c = \{v_1, \ldots, v_c\}$,
\[ \sum_{C_c} \min \deg(v_i) = O(m\alpha^{c-1}). \]

- For each peeled r-clique $R$, compute intersection of neighbors of each vertex in $R$ (= set $S$)
- Parallel for each $v$ in $S$, intersect arboricity-oriented neighbors of $v$ with $S$
  - Recurse on $S$

$\Delta = R = abe$
\[ S = N(a) \cap N(b) \cap N(e) = \{c, d, f\} \]
Theoretically: Update s-clique counts

- **Theorem**: Over all c-cliques in a graph \( C_c = \{v_1, \ldots, v_c\} \),
  \[
  \sum_{C_c} \min \\deg(v_i) = O(m\alpha^{c-1}).
  \]

- For each peeled r-clique \( R \), compute intersection of neighbors of each vertex in \( R \) (= set \( S \))
  - Parallel for each \( v \) in \( S \), intersect arboricity-oriented neighbors of \( v \) with \( S \)
    - Recurse on \( S \)

\( \triangle = R = abe \)

previous \( S = \{c, d, f\} \), \( v = c \)

\( S' = N_\rightarrow(c) \cap S = \{d\} \)
Theoretically: Update s-clique counts

**Theorem**: Over all $c$-cliques in a graph $C_c = \{v_1, \ldots, v_c\}$,
\[ \sum_{c \leq i \leq c} \min \deg(v_i) = O(m\alpha^{c-1}). \]

For each peeled $r$-clique $R$, compute intersection of neighbors of each vertex in $R$ (= set $S$)

Parallel for each $v$ in $S$, intersect arboricity-oriented neighbors of $v$ with $S$
- Recurse on $S$

\[ \triangledown = R = abe \]
new $S = \{d\}$

$r = 3$, $s = 5$
Theoretically: Update s-clique counts

**Theorem:** Over all c-cliques in a graph $C_c = \{v_1, \ldots, v_c\}$, 
$$\sum_{c_{1\leq i \leq c}} \min \deg(v_i) = O(m\alpha^{c-1}).$$

- For each peeled r-clique $R$, compute intersection of neighbors of each vertex in $R$ (= set $S$)
- Parallel for each $v$ in $S$, intersect arboricity-oriented neighbors of $v$ with $S$
  - Recurse on $S$

$r = 3$, $s = 5$

$\triangle = R = abe$

previous $S = \{d\}$, $v = d$

$S' = N_\rightarrow(d) \cap S = \emptyset$
Theoretically: Update s-clique counts

- **Theorem**: Over all c-cliques in a graph \(C_c = \{v_1, \ldots, v_c\}\),
  \[\sum_{c_{i=1}}^{c} \min \deg(v_i) = O(m\alpha^{c-1}).\]

- For each peeled r-clique \(R\), compute intersection of neighbors of each vertex in \(R\) (= set \(S\))

- Parallel for each \(v\) in \(S\), intersect arboricity-oriented neighbors of \(v\) with \(S\)
  - Recurse on \(S\)

\(r = 3, s = 5\)

This gives a 5-clique \(\{a, b, c, d, e\}\) affected by peeling \(\{a, b, e\}\)
Theoretically: Update s-clique counts

▷ Theorem: Over all $c$-cliques in a graph $C_c = \{v_1, ..., v_c\}$,
\[
\sum_{c_{1 \leq i \leq c}} \min \deg(v_i) = O(m\alpha^{c-1}). \quad [1]
\]

\[
O(m\alpha^{r-1}) = O(\alpha^{s-r-1})
\]

▷ For each peeled $r$-clique $R$, compute intersection of neighbors of each vertex in $R$ (= set $S$)

▷ Parallel for each $v$ in $S$, intersect arboricity-oriented neighbors of $v$ with $S$
  ○ Recurse on $S$

\[
= O(m\alpha^{s-2}) \text{ work}
\]

[1] Eden, Ron, Seshadhri 2020
\((r, s)\)-nucleus decomposition

- \(O(m)\) work, \(O(\log^2 n)\) span

- \(O(m\alpha^{s-2})\) work, \(O(s \log n)\) span whp

- \(O(m\alpha^{r-2} + \rho \log n)\) amortized expected work, \(O(\rho \log n)\) span whp

where \(\rho = \#\) rounds to peel entire graph

- \(O(m\alpha^{s-2})\) amortized expected work, \(O(\rho \log n)\) span whp

Direct the graph (DG) using an arboricity orientation

Count \# s-cliques per r-clique using DG

Construct a bucketing structure mapping r-cliques to a bucket based on \# s-cliques

While not all r-cliques have been peeled:
  - Peel set of r-cliques with minimum s-clique count
  - Update s-clique counts of remaining r-cliques
(r, s)-nucleus decomposition

$O(m)$ work, $O(\log^2 n)$ span $\Rightarrow$ Direct the graph (DG) using an arboricity

$O(m)$ work, $O(s)$ amortized $\Rightarrow$ Construct a bucketing structure mapping r-cliques to a bucket based on # s-cliques

While not all r-cliques have been peeled:

- Peel set of r-cliques with minimum s-clique count
- Update s-clique counts of remaining r-cliques

Practical optimizations:

- Up to a 5x speedup over our unoptimized parallel nucleus decomposition
- Up to a 2.5x reduction in space over our unoptimized parallel nucleus decomposition
Experiments
Environment

- **30-core** GCP instance (2-way hyperthreading), **240 GiB** main memory

- Used real-world Stanford Network Analysis Platform (SNAP) graphs
Comparison to other implementations

Other implementations are not theoretically efficient

- **Speedups up to 55x, median 9x** over fastest of PND, AND, AND-NN ($r = 3, s = 4$)
- **Up to 40x** self-relative speedups ($r < s \leq 7$)

- PND, AND, AND-NN have large span, are not work-efficient, or are not space-efficient (runs OOM)
Conclusion
Conclusion

▷ **Summary:**
  ○ Shared-memory parallel clustering algorithms developed with strong theoretical guarantees + practical optimizations = highly efficient and scalable implementations

▷ **Future directions:**
  ○ Dynamic nucleus decomposition
  ○ Other subgraph decompositions for other classes of graphs (e.g., bipartite graphs)
     ■ Generalization of $(\alpha, \beta)$-decomposition
Conclusion

▷ Nucleus Decomposition Github: https://github.com/jeshi96/arb-nucleus-decomp

▷ Contact me: jeshi@mit.edu
Thank you!
In practice: Keep track of r-cliques

▷ **Subgoal 1**: A way to keep track of r-cliques with min s-clique count

▷ **Julienne**: Efficient parallel bucketing structure \[1\]

▷ **Requirement 1**: Map r-cliques to unique keys

▷ **Requirement 2**: Obtain constituent r-clique vertices from keys

In practice: Keep track of r-cliques

- **Julienne**: Efficient parallel bucketing structure [1]
  - Bucket # = # of four-cliques
  - Each key in the buckets corresponds to a triangle
    - e.g., key 0 = cdg, key 1 = abe

In practice: Map r-cliques to keys

▷ An option for space savings:
▷ Two-level array and hash table:

Keys = index of r-clique in last-level tables, Values = # s-cliques

Additional optimization for cache behavior: Store last-level tables contiguously in memory
In practice: Obtain r-clique vertices from keys

Julienne:

Bucket 0: 0
Bucket 1: 2, 6, 7, 9
Bucket 2: 3, 4, 5, 8, 9, 10, 11, 12, 13
Bucket 3: 1

(a, b, c, d, e, f)

(bf, 1) (ef, 1) (bc, 2) (bd, 2) (be, 3) (cd, 2) (ce, 2) (de, 2) (ef, 1) (cd, 2) (ce, 2) (de, 1) (de, 1) (dg, 0)
In practice: Obtain r-clique vertices from keys

▷ Stored pointers:
In practice: Update s-clique counts

- Subgoal 2: A way to update s-clique counts after “deleting” r-cliques
- How do we aggregate r-cliques with updated s-clique counts in parallel?
In practice: Obtain set of updated r-cliques

▷ List buffer:

▷ Contention only when getting a new block
Other implementations are not theoretically efficient

▷ PND: Large span (> 80,000x sequential rounds compared to our alg)

▷ AND: Not work-efficient (up to 46x # of 4-cliques discovered compared to our alg)

▷ AND-NN: Not work-efficient and not space-efficient (up to 3.5x # of 4-cliques discovered compared to our alg, out of memory for skitter, livejournal, and orkut)
Comparison to other implementations

- Up to 55x speedups over PND (average 23x)
- Up to 60x speedups over AND (average 14x)
- Up to 9x speedups over AND-NN (average 3x)

- AND-NN runs out of memory on graphs with > 11 million edges

- Up to 40x self-relative parallel speedups
(r, s)-nucleus decomposition

▷ s-clique degree of a r-clique: Number of s-cliques each r-clique participates in

▷ (r, s)-nucleus decomposition: Repeatedly find + “delete” r-clique with min s-clique degree

Entire graph is in a 3-triangle-core

Entire graph is in a 2-(2, 3) nucleus
$(r, s)$-nucleus decomposition

- **s-clique degree of a $r$-clique**: Number of $s$-cliques each $r$-clique participates in
- **$(r, s)$-nucleus decomposition**: Repeatedly find + “delete” $r$-clique with min $s$-clique degree

1-$(3, 4)$ nuclei
$(r = 3, s = 4)$