Complex Network Analysis using Parallel Approximate Motif Counting

George M. Slota  Kamesh Madduri

Computer Science and Engineering
The Pennsylvania State University
gmslota@psu.edu, madduri@cse.psu.edu
fascia-psu.sf.net

IPDPS 2014
May 20, 2014
Our Contributions

- New distributed-memory parallelization of a dynamic programming-based subgraph counting scheme
  - memory-efficient
  - complementary to prior shared-memory parallelism

- Comparative network analysis using relative subgraph counts as graph signatures
  - find motifs in networks
  - detect local structure in network snapshots
  - cluster networks into categories

- Open-source tool: fascia-psu.sf.net
Talk Outline

- **Introduction**: Color-coding for subgraph counting
- **Background**: Why fast subgraph counting?
- **Algorithms**: New distributed-memory parallelization
- **Results**: Parallel performance and scalability
- **Analysis**: Large network collection using subgraph counts
Introduction

Subgraph Counting

Template
Introduction
Subgraph Counting

Template

Larger Network
Template → Larger Network
Introduction

Subgraph Counting

Template

Larger Network
Introduction
Subgraph Counting
Introduction
Color-coding [Alon et al., 1995] for Approximate Subgraph Counting

- Color-coding: randomized method to get approximate counts of tree-structured non-induced subgraphs, termed as treelets
Introduction
Color-coding [Alon et al., 1995] for Approximate Subgraph Counting

- Color-coding: randomized method to get approximate counts of *tree-structured non-induced subgraphs*, termed as *treelets*
Color-coding [Alon et al., 1995] for Approximate Subgraph Counting

- Color-coding: randomized method to get approximate counts of *tree-structured non-induced subgraphs*, termed as *treelets*
Color-coding: randomized method to get approximate counts of *tree-structured non-induced subgraphs*, termed as **treelets**

\[
\text{cnt} = 3, \quad C = 3, \quad C_{\text{colorful}} = 3!, \quad P = 3! \cdot 3
\]

Use multiple coloring iterations. Each iteration is \( O(m^2) \) work.
Introduction
Color-coding [Alon et al., 1995] for Approximate Subgraph Counting

- Color-coding: randomized method to get approximate counts of \textit{tree-structured non-induced subgraphs}, termed as \textit{treelets}
- $\text{cnt}_{\text{colorful}} = 3$, $C_{\text{total}} = 3^3$, $C_{\text{colorful}} = 3!$, $P = \frac{3!}{3^3}$
- $\text{cnt}_{\text{estimate}} = \frac{\text{cnt}_{\text{colorful}}}{P} = 13.5$

Template:

Possible Colorful Embeddings:
Color-coding: randomized method to get approximate counts of tree-structured non-induced subgraphs, termed as treelets.

\[ \text{cnt}_{\text{colorful}} = 3, \text{C}_{\text{total}} = 3^3, \text{C}_{\text{colorful}} = 3!, P = \frac{3!}{3^3} \]

\[ \text{cnt}_{\text{estimate}} = \frac{\text{cnt}_{\text{colorful}}}{P} = 13.5 \]

Use multiple coloring iterations. Each iteration is \( O(m2^k) \) work.
Motivation
Why do we want fast algorithms for subgraph counting?

- Important uses in bioinformatics, chemoinformatics, social network analysis, communication network analysis, etc.
- Counts form basis for more complex analyses:
  - Motif finding
  - Graphlet frequency distances (GFD)
  - Graphlet degree distributions (GDD) and agreements (GDDA)
  - Graphlet degree signatures (GDS)
- Counting and enumeration on large networks is very expensive, $O(n^k)$ complexity for naïve algorithm. Color-coding reduces this to $O(m2^k n_{iter})$. 
Background
Network analysis using graphlet counts

- Graphlets: all possible 2-5 vertex undirected subgraphs

Background

Network analysis using graphlet counts

- Graphlets: all possible 2-5 vertex undirected subgraphs
- Graphlet frequency distance

\[
S_i(G) = - \log \left( \frac{C_i(G)}{\sum_{i=1}^{n} C_i(G)} \right)
\]

\[
D(G, H) = \sum_{i=1}^{n} |S_i(G) - S_i(H)|
\]

Background
Network analysis using graphlet counts

- Graphlets: all possible 2-5 vertex undirected subgraphs
- Graphlet frequency distance
- Graphlet degree distribution and agreement

Background
Network analysis using graphlet counts

- Graphlets: all possible 2-5 vertex undirected subgraphs
- Graphlet frequency distance
- Graphlet degree distribution and agreement

\[
S^j_G(k) = \frac{d^j_G(k)}{k}
\]

\[
N^j_G(k) = \frac{S^j_G(k)}{\sum_{k=1}^{\infty} S^j_G(k)}
\]

\[
A^j(G, H) = 1 - \frac{1}{\sqrt{2}} \left( \sum_{k=1}^{\infty} [N^j_G(k) - N^j_H(k)]^2 \right)
\]

Background
Network analysis using graphlet counts

- Graphlets: all possible 2-5 vertex undirected subgraphs
- Graphlet frequency distance
- Graphlet degree distribution and agreement
- Graphlet degree signature

\[ S_i(u, v) = 1 - w_i \times \frac{|\log(u_i + 1) - \log(v_i + 1)|}{\log(\max\{u_i, v_i\}) + 2} \]

Background
Network analysis using treelet counts

- Can we use treelets instead of graphlets? Are they more/less powerful?
- Goals of this work:
  - Create distributed-memory subgraph counting program to produce counts on larger networks, and faster counts on smaller networks.
  - Quantitative analyses using GFD and GDD with treelets to evaluate efficacy.
  - Evaluate effect of noise on treelet counts by deleting vertices and edges, as well as rewiring edges in various networks.
FASCIA
Fast Approximate Subgraph Counting (for In-memory/Insightful/I* Analytics)

- Previous work: FASCIA for shared-memory color-coding treelet counting [Slota and Madduri, 2013].
  - Memory reduction through efficient table and color set representations
  - Work reduction through template partitioning
  - Multilevel algorithm parallelization
- Present work: FASCIA for distributed memory
  - Further memory and communication reductions through CSR-like representation of table
  - Partitioned counting allowing counts for larger networks
  - Distributed counting for faster counts on smaller networks
1: Partition input template $T$ ($k$ vertices) into subtemplates $S_i$ using single edge cuts.

2: Determine $N_{iter} \approx \frac{e^k \log 1/\delta}{\epsilon^2}$, the number of iterations to execute.

$\delta$ and $\epsilon$ are input parameters that control approximation quality.

3: for $it = 1$ to $N_{iter}$ do

4: Randomly assign to each vertex $v$ in graph $G$ a color between $0$ and $k - 1$.

5: Use a dynamic programming scheme to count colorful non-induced occurrences of $T$.

6: Take average of all $N_{iter}$ counts to be final count.
Each task gets a subset of $v \in G$, counts for this subset are further computed in parallel for each task.

- Semi-partitioned: Each task holds full table for child subtemplates.

```plaintext
for $it = 1$ to $Niter$ do
    Color $G(V, E)$ with $k$ colors
    for all subtemplates $S_i$ in reverse order of partitioning do
        Init $Table_{i,d}$ for $V_d$ (vertex partition on task $d$)
        for all $v \in V_d$ do in parallel
            for all $c \in C_i$ do
                Compute all $Count_{S_i,c,v}$
        end for all $v \in V_d$ do
        end for all subtemplates $S_i$ do
        $N_d, I_d, B_d \leftarrow$ Compress($Table_{i,d}$)
        for all $d = 1$ to $NumTasks$ do
            $N_i, I_i, B_i \leftarrow$ Bcast($N_d, I_d, B_d$)
        end for all $d = 1$ to $NumTasks$ do
        $Count_d += \sum_{V_d} \sum_{C_T} Count_{T,c,v}$
    end for all subtemplates $S_i$ do
    $Count \leftarrow$ Reduce($Count_d$)
    Scale $Count$ based on $Niter$ and colorful embed prob.
```
Dynamic programming table corresponding to each subtemplate can be represented as a $n \times C_i$ rectangular matrix ($n$: number of graph vertices, $C_i$: number of possible color sets of subtemplate $i$).

We represent the table in a CSR-like format using three arrays:
- $N$: count values in table
- $I$: color set indexes
- $B$: offsets for each vertex
for all $it = 1$ to $Niter$ in parallel do
    Color $G(V, E)$ with $k$ colors
    Initialize 3D count table
    for all $S_i$ in reverse order of partitioning do
        for all $v \in V$ in parallel do
            Update count table for template $S_i$
            using child subtemplate counts

    ▶ MPI task-level
    ▶ multithreaded
Results

Experimental setup - systems and graphs

- Performance results on Compton, a Sandia cluster with dual-socket Intel Xeon E5-2670 (Sandy Bridge) nodes and 64 GB main memory per node.
- Studies also performed on Cyberstar and Hammer clusters at Penn State.
- Networks from SNAP, Konect, DIMACS, UF sparse matrix, and Virginia Tech NDSSL collections; GTgraph and igraph generators.

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Count</th>
<th>$n \times 10^3$</th>
<th>$m \times 10^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>Collaboration</td>
<td>6</td>
<td>26</td>
<td>425</td>
</tr>
<tr>
<td>Communication</td>
<td>4</td>
<td>30</td>
<td>63</td>
</tr>
<tr>
<td>$G(n, p)$</td>
<td>4</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Peer-to-peer</td>
<td>9</td>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>Bio PPI</td>
<td>4</td>
<td>0.7</td>
<td>22</td>
</tr>
<tr>
<td>Road</td>
<td>5</td>
<td>440</td>
<td>1970</td>
</tr>
<tr>
<td>Scale-free</td>
<td>4</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Social</td>
<td>6</td>
<td>60</td>
<td>150</td>
</tr>
<tr>
<td>Small-world</td>
<td>4</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Web Crawl</td>
<td>4</td>
<td>280</td>
<td>875</td>
</tr>
<tr>
<td>Orkut</td>
<td>-</td>
<td>3100</td>
<td>-</td>
</tr>
<tr>
<td>Portland</td>
<td>-</td>
<td>1620</td>
<td>-</td>
</tr>
</tbody>
</table>
Results
Experimental setup - templates used

- UX-1 indicates chain or simple path, UX-2 indicates more complex tree
- T7-X indicates all 7 vertex treelets, used for evaluating effects of network noise on relative counts
Parallel speedup on 15 nodes; 3.5× (U12-1) and 7× (U12-1) for Orkut crawl
Results
Running times and parallel scaling

- Parallel speedup on 15 nodes: $3.5 \times (U12-1)$ and $7 \times (U12-1)$ for Orkut crawl
- Communication time is about 15-25% of total time
Results
Running times and parallel scaling

- Parallel speedup on 15 nodes; $3.5 \times (U12-1)$ and $7 \times (U12-1)$ for Orkut crawl
- Communication time is about 15-25% of total time
- Near-linear speedup for distributed counting on C.elegans PPI network

![Image showing execution time (s) against # MPI tasks for different templates U12-1 and U12-2]
Communication volume reduction

- Maximal data transfer during counting on Orkut for templates from 7 to 12 vertices
- About 35% mean reduction in transfer costs with CSR compression
Results

Graphlet frequency distance analysis

- Treelet frequency distance, GFD calculated with counts of all 4-9 vertex treelets (92 total)
- Intra-class mean agreements highest for 5/10 classes
Results
Graphlet degree distribution agreement analysis

- Treelet degree distribution agreements, GDDA calculated with counts of orbits of all 3-7 vertex treelets (83 total)
- Similar observations as with GFD, intra-class mean highest 6/10 classes
Results

Vertex deletion

- 5, 10, 20, 50, and 75% vertices deleted, evaluated based on GFD for only 7 vertex templates, relative counts shown
- Largest disagreement for Notre Dame webcrawl, value of 4.1
- Mean disagreement between networks is 9.2

![Graphs showing relative counts for different modifications and networks]
5, 10, 20, 50, and 75% edges deleted
- Max disagreement value of 1.2 with Gnutella p2p snapshot
- Demonstrated relative treelet counts are useful analytic even on networks with a relatively high proportion of known vertices to known edges (e.g. PPI networks)
5, 10, 20, 50, and 75% edges rewired
6.6 and 10.4 disagreements with Slashdot and Notre Dam webcrawl
Minimal change with random and Gnutella networks; less inherent structure?
Conclusions

- Partitioned subgraph counting (i.e., partitioned dynamic programming table) makes it feasible to analyze large-scale networks on clusters.
- Distributed subgraph counting accelerates subgraph counting for small networks.
- Treelets counts seem to be as powerful as graphlets for network analysis.
- Relative treelet counts more sensitive to vertex sampling than edge sampling.


Acknowledgments

- NSF grant ACI-1253881
- Penn State systems, NSF grant OCI-0821527
- NSF XSEDE, supported by NSF grant OCI-1053575
- Sandia National Laboratories, for access to the Compton system
  - Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.
Questions?

- fascia-psu.sf.net
- George M. Slota (gms5016@psu.edu)
- Kamesh Madduri (madduri@cse.psu.edu)