Parallel Algorithms for Counting Triangles and Computing Clustering Coefficients

S M Arifuzzaman, Maleq Khan and Madhav V. Marathe
Virginia Bioinformatics Institute at Virginia Tech, Blacksburg, VA 24061

The Problem and Contributions

We present MPI-based parallel algorithms for counting triangles and computing clustering coefficients in massive networks.

❑ A triangle in a graph G(V, E) is a set of three nodes u, v, $w \in V$ such that there is an edge between each pair of nodes. The number of triangles incident on node v, with *adjacency list* N(v), is defined as,

$$T_v = |\{(u,w) \in E \mid u, w \in N(v)\}|$$

Counting triangles is important in the analysis of various networks, e.g., social, biological, web etc. Emerging massive networks do not fit in the main memory of a single machine and are very challenging to work with. Our distributed-memory parallel algorithm allows us to deal with such massive networks in a time- and space-efficient manner. We were able to count triangles in a graph with 2 billions of nodes and 50 billions of edges in 10 minutes.

❑ The clustering coefficient (CC) of a node $v \in$V with degree d_v is defined as,

$$C_v = \frac{2T_v}{d_v(d_v - 1)}$$

Computing clustering coefficients is also an important problem which is almost equivalent to counting triangles.

❑ We also show how edge sparsification [1] can be used with our parallel algorithm to find approximate number of triangles. Our parallel adoption of sparsification technique improves the accuracy over the original sequential algorithm in [1].

❑ In addition, we propose a simple modification of a state-of-the-art sequential algorithm that improves both runtime and space requirement.

Improved Sequential Algorithm

Many algorithms use adjacency matrix representation which is not suitable for large graphs as it takes $O(n^2)$ memory. NodeIterator++ [2,3,5] is a state-of-the art algorithm that uses adjacency list representation.

NodeIterator++ uses an ordering, $\prec$, of nodes to avoid duplicate count of triangles. A degree-based ordering, shown below, reduces running time significantly comparing to an arbitrary ordering (details are in our technical report [4]).

$$u \prec v \Leftrightarrow (d_u < d_v) \vee (d_u = d_v \wedge u < v)$$

Proposed Modified Algorithm: NodeIteratorN

Unlike NodeIterator++, our algorithm NodeIteratorN performs comparison $u \prec v$ for each edge $(u,v) \in E$ in preprocessing step rather than doing same in computing step. NodeIteratorN reduces memory consumption by half, and improves running time as shown below in the table.

The pseudocode of the algorithm is given in the right. $N(v)$ stores a subset of the neighbors of node v.

```
{Preprocessing: first for loop}
for each edge $(u,v) \in E$ do
  if $u \prec v$ then
    store $v$ in $N(u)$
  else
    store $v$ in $N(v)$
for $v \in V$ do
  sort $N(v)$ in ascending order
$T$=0        //counts of triangles
for $v \in V$ do
  for $u \in N(v)$ do
    $S = N(v) \cap N(u)$
    $T = T + |S|$
```

Table: (left) Runtime comparison of NodeIterator++ (N++) and NodeIteratorN (NN); (right) Dataset used for experiments; Hcc(n,d), an artificially generated network with n nodes and d avg. degree, has high triangle density.

Networks	Runtime(sec)		No. of triangles
	N++	NN	
Email-Enron	0.08	0.03	0.7M
web-BerkStan	3.3	0.4	64.7M
LiveJournal	40.35	13	285.7M
Miami	43.56	16	332M
Gnp(50K, 20)	0.14	0.06	1290
Gnp(500K, 20)	1.8	0.6	1308
Hcc(5M, 50)	40	8	1.5B
Hcc(15M, 90)	553	52.6	15B
Hcc(5M, 200)	931	114	24.7

Networks	Nodes	Edges
Email-Enron	37K	0.36M
web-BerkStan	0.69M	13M
Miami	2.1M	100M
LiveJournal	4.8M	86M
Twitter	42M	2.4B
Gnp(n,d)	n	0.5nd
Hcc(n,d)	n	0.5nd

Parallel Algorithm for Triangle Counting

With P processors, the graph is partitioned into P partitions. Each processor reads its own partition in parallel from the input file. Each processor performs local computation and results are then combined.

Partitioning

❑ Each processor works on $G_i(V_i,E_i)$, a subgraph of the original graph G(V,E) induced by V_i.

❑ V is partitioned into sets of core vertices V_i^c ($0 \le i < P$), each with equal number of vertices, such that, for any two processors i and j, $V_i^c \cap V_j^c = \emptyset$ and $\cup_i V_i^c = V$

❑ V_i contains a set of core vertices V_i^c and some extra vertices- neighbors of core vertices; E_i contains all the edges between any two vertices of V_i.

Counting Triangles

Each processor i counts total triangles incident on $v \in V_i^c$. Pseudocode for the overall parallel algorithm and CountTriangle routine are provided below.

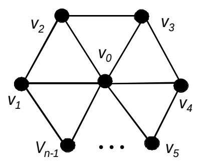
```
{T: count of triangles}
for each processor i, in parallel, do
  $G_i(V_i, E_i)$ = ReadGraph(G, $i$)
  $T$ = CountTriangles($G_i$, $i$)
MpiBarrier
MpiReduce($T$)
```

```
{T: count of triangles}
for $v \in V_i$ do
  sort $N(v)$ in ascending order
$T$=0
for $v \in V_i^c$ do
  for $u \in N(v)$ do
    $S = N(v) \cap N(u)$
    $T = T + |S|$
```

Load-Balancing

❑ We assign equal number of core vertices per processor. But, load can be imbalanced if the network has skewed degree distribution.

❑ However, degree-based ordering provides very good load balancing without additional overhead. Consider the example network shown in the right. Although v_0 has degree n-1, we have $N(v_0)$ = 0 and $N(v_i)$ <= 3, for all i, with degree-based ordering.



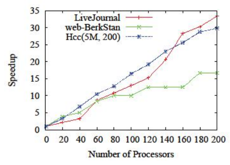
Performance

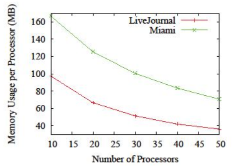
Our parallel algorithm

❑ scales very well with size of networks and number of processors.

❑ is significantly faster than the only known distributed-memory parallel algorithm by Suri et al., 2011 [3], to the best of our knowledge.

❑ is able to run on a network with 2B nodes and 50B edges in 10 minutes, whereas, the largest network processed by [3] is a network with 42M nodes and 2.9B edges.





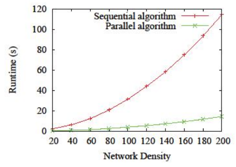


Figure (right): (top) strong scaling, (middle) memory scalability and (bottom) runtime vs network density

Table: Runtime comparison: our algo. with [3]

Networks	Number of triangles	Runtime	
		Our Algo.	[3]
web-BerkStan	65M	0.03s	1.7m
LiveJournal	285.7M	0.39s	5.33m
Twitter	34.8B	15m	423m
Hcc(5M, 200)	24.7B	3.86s	-
Hcc(2B, 20)	90B	90s	-
Hcc(2B, 50)	600B	9m	-
Miami	332M	0.5s	-

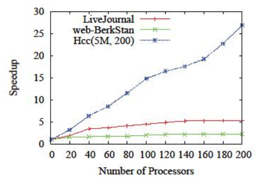
Parallel Computation of Clustering Coefficients

❑ For computing CC of a node $v \in V$, we need T_v, the number of triangles incident on v. Partial count of triangles of v may reside at different processors which needs to be aggregated.

❑ If we use n element count array for n nodes at each processor, then we can use MPI_Reduce to sum up those counts easily. But, it requires O(n) memory per processor.

❑ We employ external memory aggregation: each processor i writes P intermediate disk files F_j each for one distinct processor j with counts of triangles found for $v \in V_j^c$; all F_js are then aggregated by processor j. Each processor j computes C_v for all $v \in V_j^c$.

```
{T: array of counts of local triangles}
for $v \in V_i$ do
  $T_v = 0$
for $v \in V_i^c$ do
  for $u \in N(v)$ do
    $S = N(v) \cap N(u)$
    $T_v = T_v + |S|$
    $T_u = T_u + |S|$
    for $w \in S$ do
      $T_w = T_w + 1$
Communicate $T_v$ for each $v \in V_i - V_i^c$
Aggregate $T_v$ for each $v \in V_i^c$
Compute $C_v$ for each $v \in V_i^c$
```



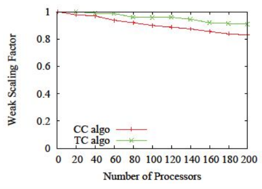


Figure: (left) Pseudocode of the algorithm; (upper right) strong scaling; and (bottom right) weak scaling of clustering coefficient and triangle counting algorithm.

Parallel Algorithms for Approximate Triangle Counting

❑ A sparsification technique used in [1] works as follows: each $(u,v) \in E$ is selected with a probability p and discarded with a probability 1-p. Let T_s be the number of triangles in the sparsified graph, then the estimated number of triangles in G is $1/p^3 * T_s$. The estimator is unbiased, because

$$\mathbf{E}[1/p^3 * T_s] = T$$

❑ In our parallel algorithm, each processor i sparsifies its own subgraph $G_i(V_i,E_i)$ independently. Note that, an edge that overlaps in two partitions can survive in one partition, but not in the other. This independence improves accuracy of the estimation (see [4] for details).

Table: Accuracy and speedup of the algorithm while running on LiveJournal graph

p	0.1	0.2	0.3	0.4	0.5
Accuracy	99.61	99.685	99.832	99.898	99.947
$1/P^2$	100	25	11.1	6.25	4
Speedup	35.3	17.6	8.1	5	3.16

Table: A comparison of variance and average error of our algorithm with [1]

Networks	Variance		Avg Error (%)		Max Error (%)		Actual Count
	Our Algo.	[1]	Our Algo.	[1]	Our Algo.	[1]	
web-BerkStan	1.287	2.027	0.389	0.392	1.02	1.08	64.7M
LiveJournal	1.77	1.958	1.46	1.86	3.88	4.75	285.7M

Reference

[1] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "DOULION:counting triangles in massive graphs with a coin," in *Proc. of the 15th KDD*, 2009, pp. 837–846.
[2] T. Schank and D. Wagner, "Finding, counting and listing all triangles in large graphs, an experimental study," in *Proc. of the 4th Intl. Conf. on Experimental and Efficient Algorithms, 2005, pp. 606–609.*
[3] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in *Proc. of the 20th Intl. Conf. on World Wide Web (WWW)*, 2011.
[4] Tech. Report, Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA, No. NDSSL 12-042, July 2012. Available online: http://staff.vbi.vt.edu/maleq/papers/clusterco-TR-12-042.pdf
[5] M. Latapy, "Main-memory triangle computations for very large (sparse(power-law)) graphs," *Theor. Comput. Sci., vol. 407, pp. 458–473, 2008.*