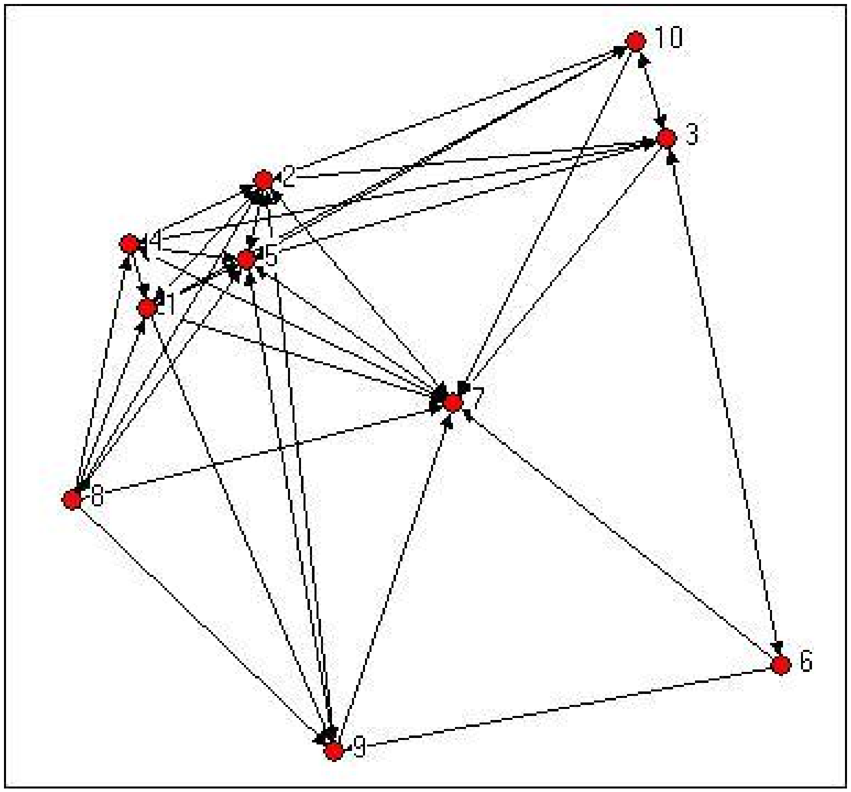
Parallel Clustering Coefficient Calculation (Triangle Counting) in Sparse Graphs



Ishan Chawla (0029031621)

Kapil Earanky (0029007743)

Problem Statement

1. Finding triangles in massive graphs is a fundamental problem, which has received much attention recently because of its importance in graph analysis.
2. For example, in social network graphs a triangle indicates a group of friends who are also mutual friends with each other. Triangles also act as measures of transitivity ratio and the clustering coefficients of a graph.
3. The typical algorithm used for counting triangles in a small graph, involves matrix-matrix multiplication, with a complexity of $O(n^3)$. Therefore, as networks become larger and complex, it becomes necessary to look at parallel algorithms which are more scalable and memory-efficient.

Literature Review

1. PATRIC: A Parallel Algorithm for Counting Triangles in Massive Networks

2. Counting Triangles and the Curse of the Last Reducer

3. Main-memory Triangle Computations for Very Large (Sparse (Power-Law)) Graphs

4. A Space-efficient Parallel Algorithm for Counting Exact Triangles in Massive Networks

Contributions

1. We implemented the compact forward algorithm proposed by Latapy et al. [1] using MPI and measured parallel runtime, speedup, scaled speedup and efficiency. We also propose a few novel optimisations to the algorithm.
2. We investigated the effect of different graph partitioning techniques on the PATRIC triangle counting algorithm using metrics proposed by Arifuzzaman et al. [2] and compare the metrics in terms of performance.

[1] M. Latapy, Main-memory triangle computations for very large (sparse (power-law)) graphs. Theor Comput Sci, 407 (2008), pp. 458–473

[2] Shaikh Arifuzzaman , Maleq Khan , Madhav Marathe, PATRIC: a parallel algorithm for counting triangles in massive networks, Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, October 27-November 01, 2013, San Francisco, California, USA

Compact-Forward Algorithm

Algorithm 7 – *compact-forward.* *Lists all the triangles in a graph.*

Input: the adjacency array representation of G
Output: all the triangles in G

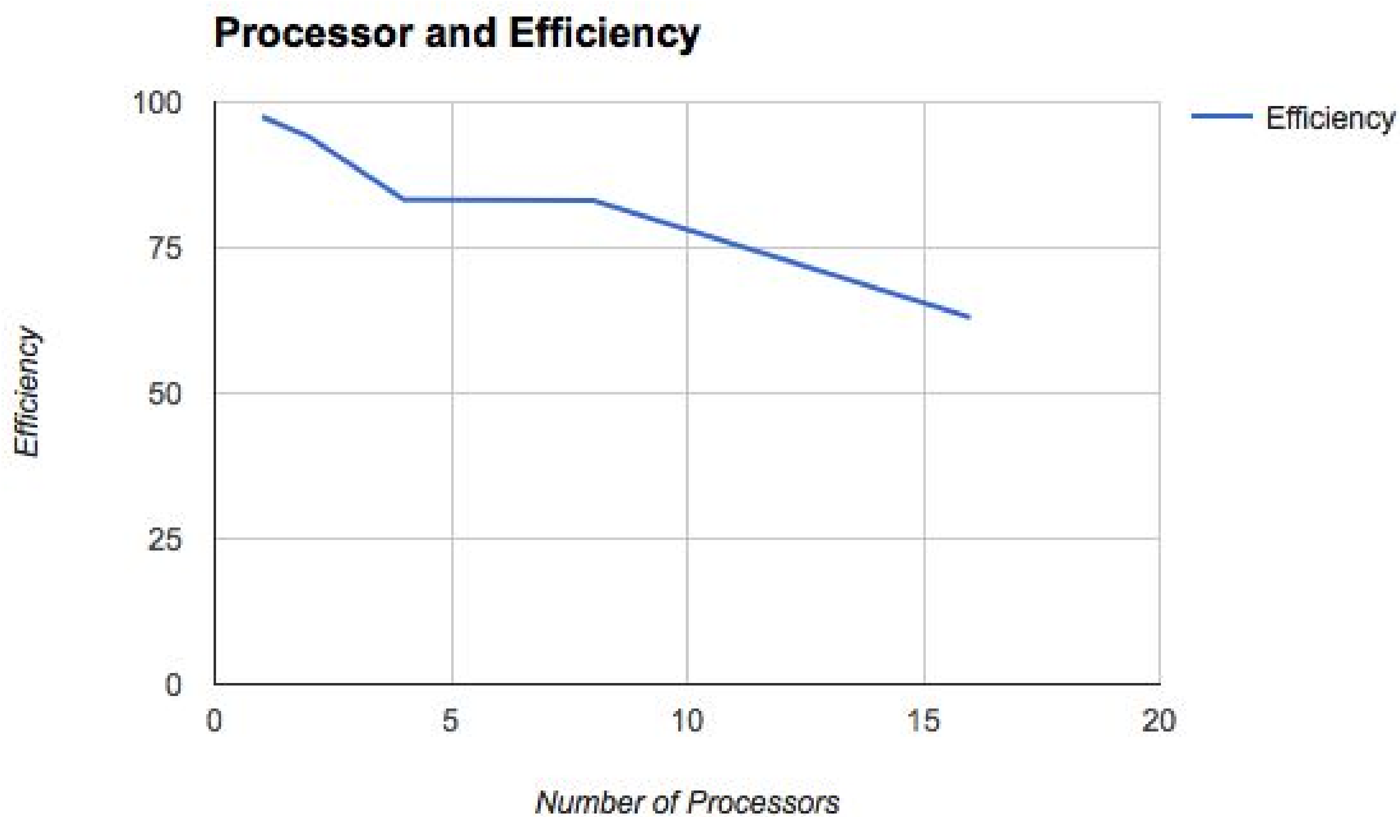
1. number the vertices with an injective function $\eta()$
   such that $d(u) > d(v)$ implies $\eta(u) < \eta(v)$ for all u and v
2. sort the adjacency array representation according to $\eta()$
3. for each vertex v taken in increasing order of $\eta()$:
  3a. for each $u \in N(v)$ with $\eta(u) > \eta(v)$:
    3aa. let u' be the first neighbor of u, and v' the one of v
    3ab. while there remain untreated neighbors of u and v and $\eta(u') < \eta(v)$ and $\eta(v') < \eta(v)$:
      3aba. if $\eta(u') < \eta(v')$ then set u' to the next neighbor of u
      3abb. else if $\eta(u') > \eta(v')$ then set v' to the next neighbor of v
      3abc. else:
        3abca. output triangle $\{u, v, u'\}$
        3abcb. set u' to the next neighbor of u
        3abcc. set v' to the next neighbor of v

Methodology

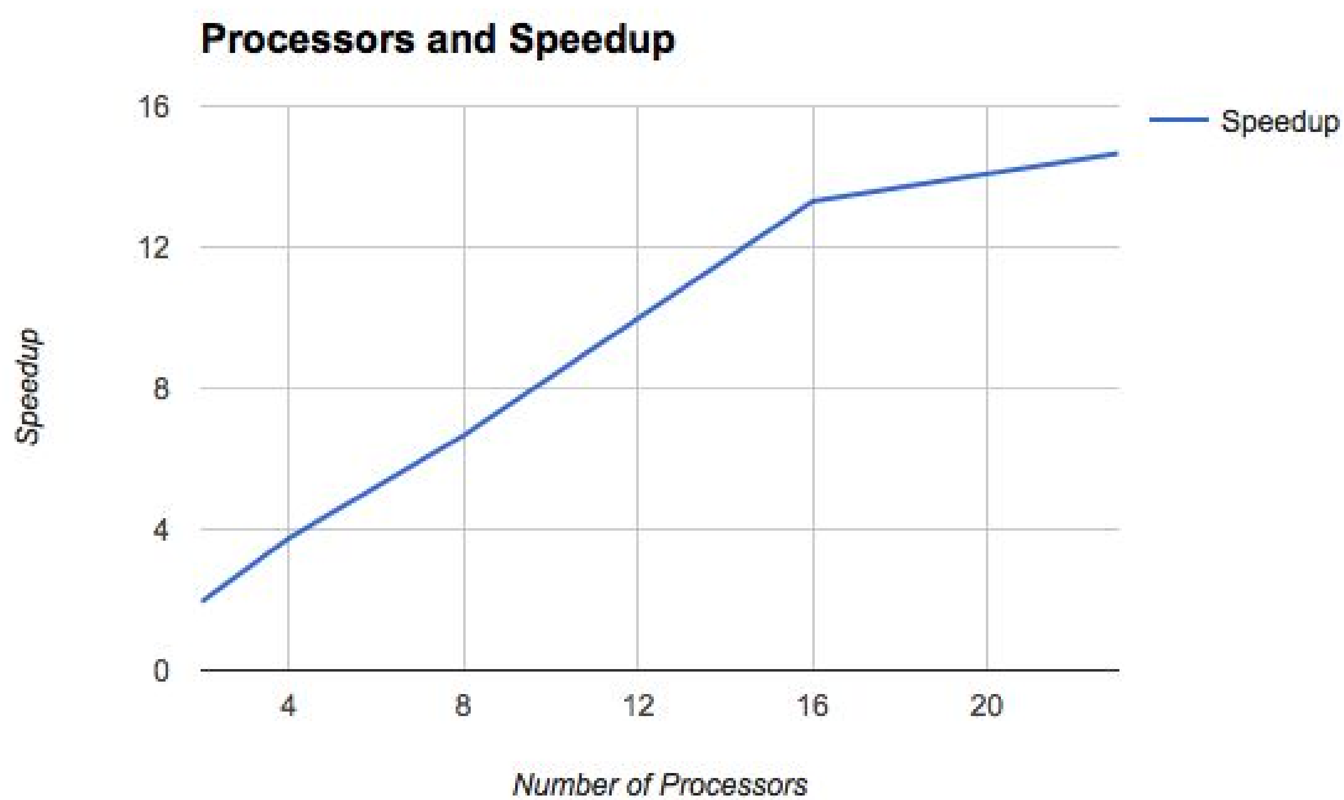
1. This algorithm counts the exact number of triangles, as opposed to an approximate or a multiplicative factor of 3 or 6 off the triangles computed by some other algorithms.
2. As the graphs are sparse , we use adjacency lists as the core data structure to save space.
3. We choose the division according to vertices and assign them in a block manner. We choose the vertices each processor reads as implicitly calculable from the processor rank. Hence a one to all broadcast of the boundaries is not required as in some previous work
4. So the first rank processor gets the first chunk of work and so on.

Implementation Details

1. One of the problems we faced was how to remap the graph according to the degrees so that the isomorphic graph generated had vertices numbered in decreasing order of degree.
2. The structure in the algorithm allowed each processor to read a non overlapping partition of the graph from a file and then remap it so that the highest degree vertices had the least numbering.
3. For finding triangles, each node would need the adjacency list of the neighbours of the node it is processing. Since this communication takes place in parallel and if there are not major skews in the graphs we observed that , it is not a major contribution to the run time.
4. After implementing the compact forward on each node, an all to all reduce was used to calculate the number of triangles.

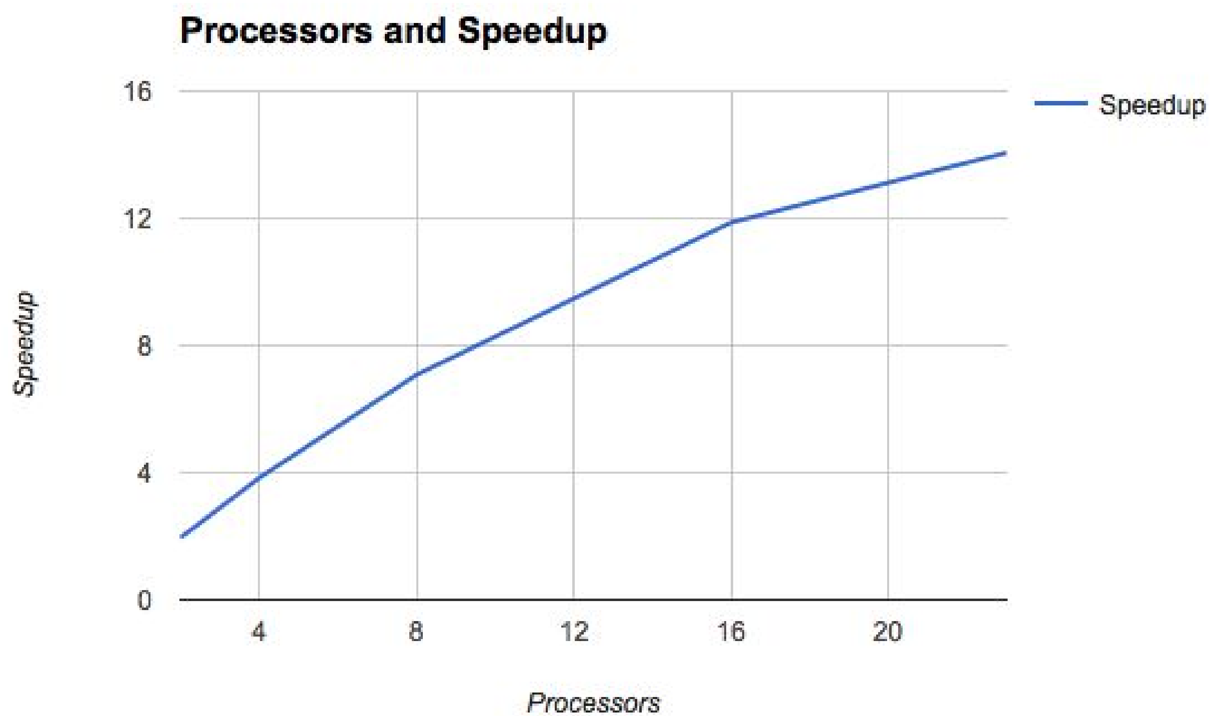


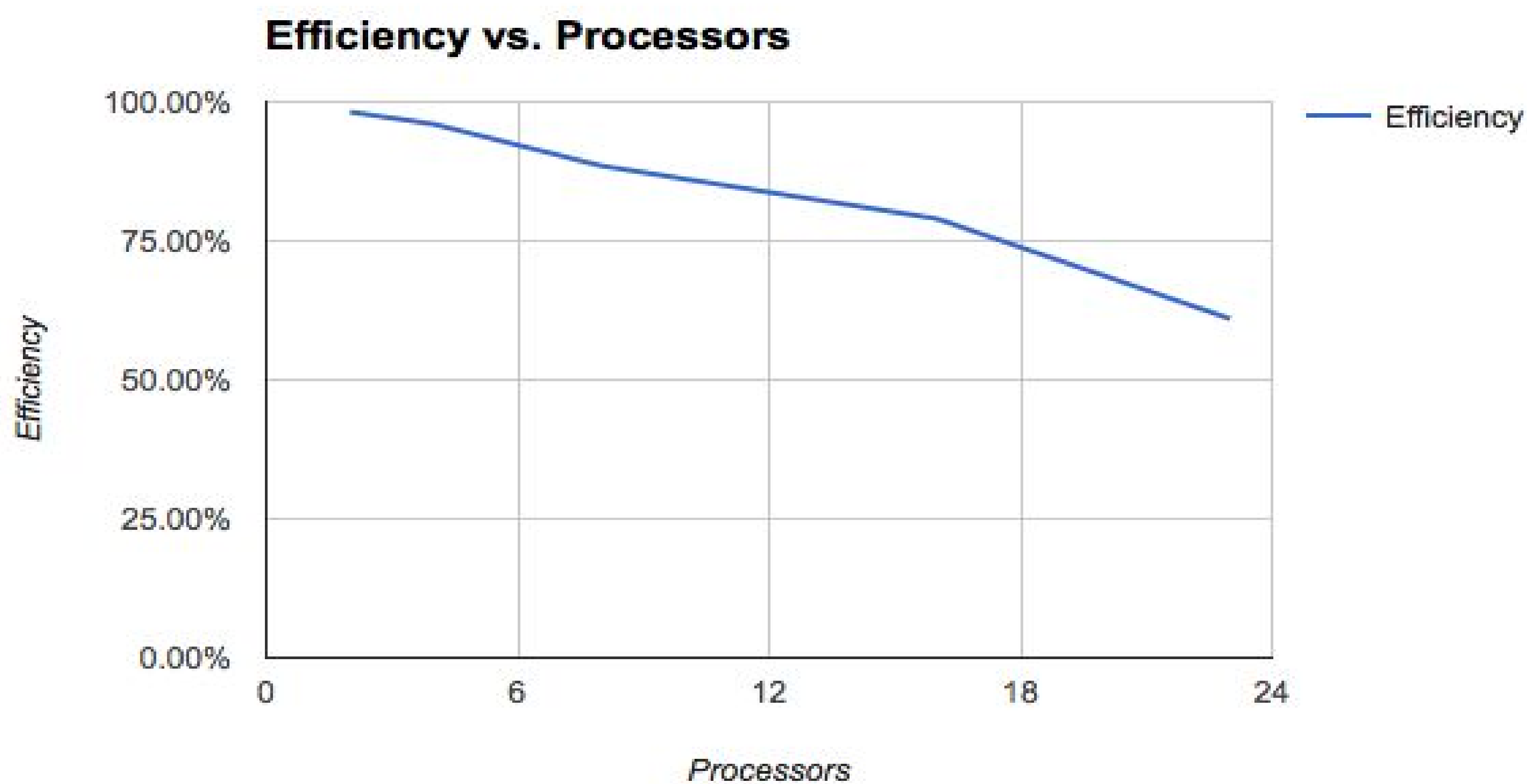
For 100000 , number of triangles are 11,000



For 100000 vertices and 11000 triangles

For 5000000 vertices and 5,50,000 triangles




Efficiency vs. Processors
100.00%
75.00%
50.00%
25.00%
0.00%
Efficiency
0
6
12
18
24
Processors
Efficiency


Scaled speedup, Speedup and IdealSS
10
7.5
5
2.5
0
2
3
4
5
6
7
8
Processors
Scaled speedup
Speedup
IdealSS

Effects of graph partitioning on PATRIC triangle counting algorithm

1. As the size and complexity of graphs increases, it becomes essential to take into account the structural properties of the graph in order to distribute work evenly across all processors.
2. We tried partitioning the graph using metrics proposed as part of the PATRIC paper. The metrics are functions f, which return an integer value for each node.

$$f : V \rightarrow N$$

3. The task of partitioning then involves assigning nodes v_p to processor p such that $\Sigma_{v \text{ in } vp} f(v) \sim \Sigma_{v \text{ in } V} f(v)/p$

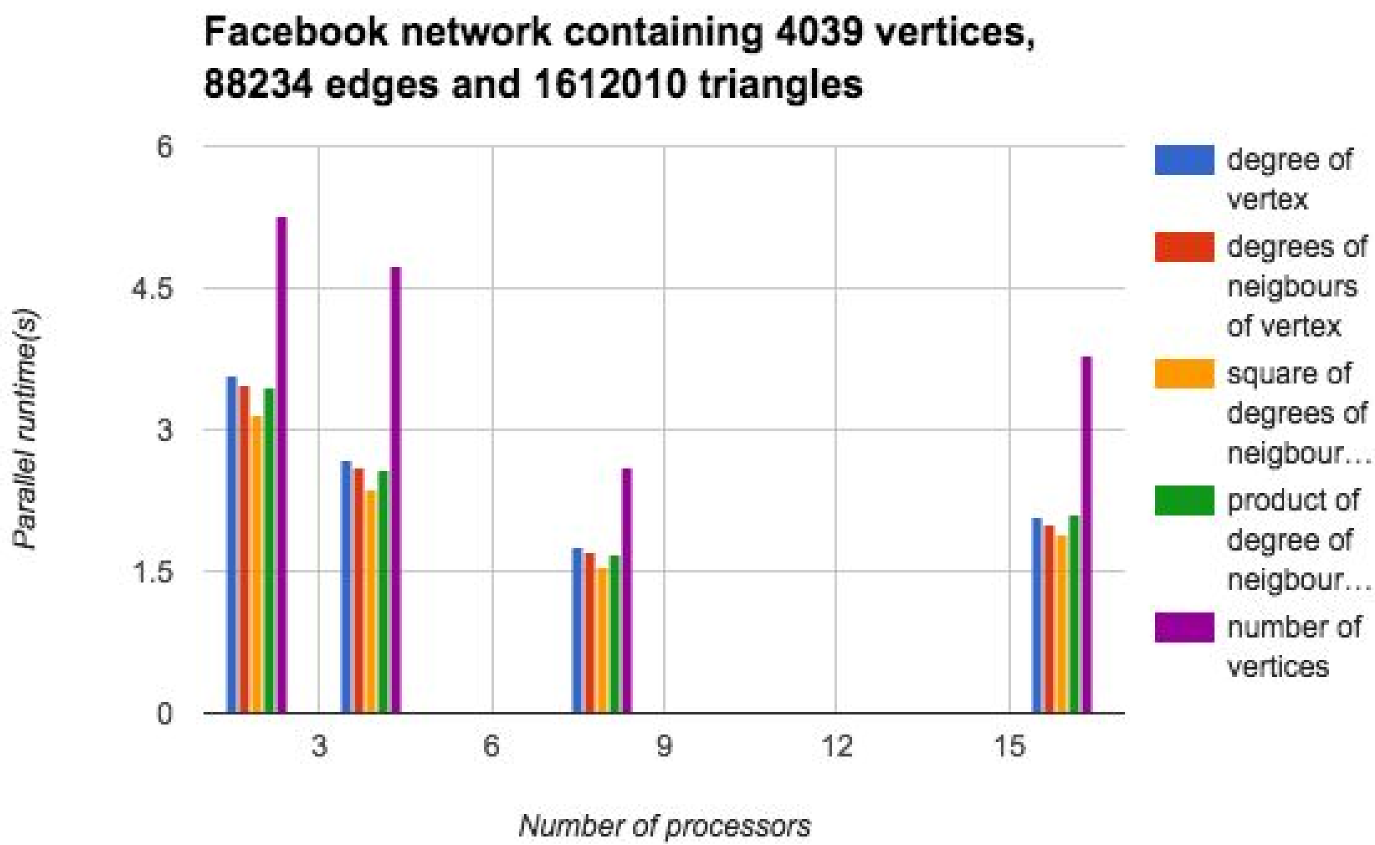
Partitioning Metrics

Name of metric	Function
Degree of vertex	$f(v) = d_v$
Degree of neighbors of vertex	$f(v) = \Sigma d_u \ \forall (v,u)$ in E
Square of degree of neighbors of vertex	$f(v) = \Sigma d_u^2 \ \forall (v,u)$ in E
Product of degree of neighbour and vertex	$f(v) = \Sigma d_u d_v \ \forall (v,u)$ in E
Number of vertices	$f(v) = 1$

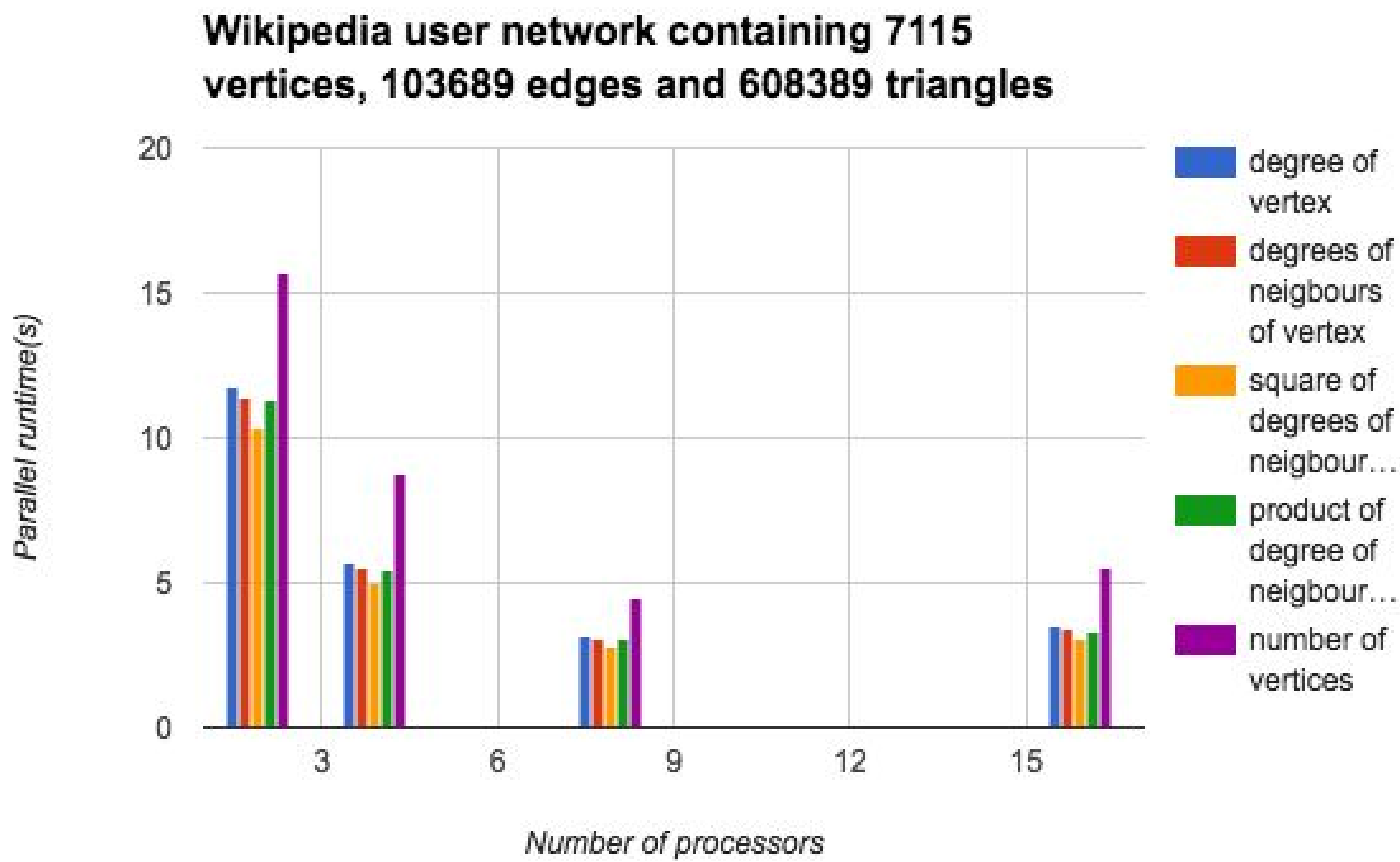
Implementation Details

1. Since we’re dealing with very large graphs, the measurement of partitioning metrics must themselves be done in parallel on each processor in order to avoid partitioning itself becoming a bottleneck.
2. Each processor reads a chunk of the graph assigned to it and computes the partitioning metrics for its own local chunk. It’s important to note here that some metrics, such as *degree of neighbors of vertex* may require data from nodes belonging to different processors. Each processor accumulates all such requests for each other processor and sends them all at once in order to avoid extra communication.
3. Once the partitioning is done, we use the PATRIC triangle counting algorithm to compute the number of triangles in the graph.

Performance comparison of different metrics



Performance comparison of different metrics



Results

1. We generated sparse graphs with up to 5,000,000 vertices and edges linear in the number of vertices. The graphs had a predetermined number of triangles.
2. We then performed experiments to calculate strong scaling characteristics, scaled speedups, speedups and performance from our implementation of the compact forward algorithm.
3. We observed speedups between the ranges of 3 and 15 using up to 16 processors. The speedups were very close to the maximum speedup possible (calculated from Amhdahl’s law).
4. We also observed that the scaled speedup curve from our implementation is approximately linear in the number of processors.

Results (contd.)

5. Further, we investigated the effects of graph partitioning techniques on the performance of the PATRIC triangle counting algorithm. We tested five partitioning metrics - *degree of vertex*, *degree of neighbors of vertex*, *square of degree of neighbors of vertex*, *product of degree of neighbour and vertex*, *number of vertices*.
6. We noticed that the *square of degree of neighbors of vertex* metric performed best on the graphs we tested, namely - Social circles: Facebook and Wikipedia vote network. We observed an improvement in performance of around 10-15% from partitioning the graph using metrics based on the degree of neighbouring vertices over edge based partitioning techniques used typically.

Conclusions and Future Work

1. The novelty of compact forward algorithm lies in its minimal serial component. Communication between processors occurs only when
    a. Exchanging subsets of adjacency lists.
    b. Reduce operation for counting the number of triangles.
2. A possible optimization when there are popular nodes with several connections in the graph would be to use a hybrid edge and vertex partitioning approach whereby the popular vertex’s edges are split between multiple processors.
3. Also neighbours of such popular vertices could replicate the adjacency list thereby handling traffic more efficiently.
4. Finally, degree based partitioning metrics work very well for the triangle counting problem.