Frontiers in Computational Science and Engineering
Vol. 2, No. 1, Jan - Mar 2026, pp. 11-19
ISSN: 3107-3654 (O), DOI: https://doi.org/10.17051/FCSE/02.01.02

Scalable Parallel Graph-Based Algorithms for Complex Network Simulations

P. Sedhupathy

Asst Professor, Department of Computer Science (Artificial Intelligence & Data Science),
Dr. SNS Rajalakshmi College of Arts and Science, Coimbatore, Email: sedhupathy@gmail.com
https://orcid.org/0009-0009-2947-6540

Article Info

Article history:

Received : 07.11.2025
Revised : 09.12.2025
Accepted : 05.01.2026

Keywords:

scalable computing,
parallel graph algorithms,
complex network simulation,
high-performance computing,
distributed systems,
graph partitioning,
load balancing,
parallel efficiency,
strong and weak scalability,
communication overhead.

ABSTRACT

Simulations on large-scale complex networks underpin the study of systems within social interactions, biological processes, and infrastructure networks. Nevertheless, traditional graph algorithms are frequently difficult to operate at scale, as they need greater amounts of data, have irregular graph connectivity, and in parallel settings, encounter significant communication costs. The paper discusses these issues through the development of a scalable parallel graph based algorithm that can be used to efficiently simulate complex networks. The approach takes advantage of distributed and hybrid parallel computing systems, which combine message passing (MPI) with shared-memory algorithms (OpenMP/GPU acceleration) to balance computation and communication. They use a partitioning strategy to evenly distribute workload and reduce inter-processor communication and an adaptive load balancing mechanism to better utilize the resources. Experimental analysis on both large synthetic and real-world systems shows that the performance is vastly improved as compared to those of existing methods. The algorithm suggested demonstrates significant speedup as the number of processors increase, high parallel performance and has good and bad scalability with respect to the size of the graph. Optimized data exchange strategies aid in minimizing communication overhead, which contributes to the overall system performance. The findings bring out the efficiency of the suggested method within the large scale graph processing and hence it suits real-time and high-performance simulations of networks. The contribution of this work to the development of scalable graph processing methods is that it offers a foundation to further studies in parallel and distributed complex network analysis.

1. INTRODUCTION

Complex networks are very popular in modeling real world systems such as, social interactions, biological processes, transportation infrastructures, and communication networks. The natural representation of these systems is in the form of large scale graphs, with nodes and edges reflecting the rich set of relationships and dynamics involved. As data volume grows exponentially, the new generation of applications needs scalable and efficient algorithms that are able to handle huge graphs. Nevertheless, conventional graph algorithms can frequently cope with such scale because of the amount of computation required and the limits of memory.

The solution to these restrictions is to utilize scalable graphs-based and parallel algorithms that have been studied widely. Graph partitioning is used and clustering techniques are enhanced to enhance workload allocation and improve inter-process communication [1] -[4]. Further advanced partitioning algorithms like multilevel and hypergraph algorithms provide even more scalability and efficacy in a large network [12], [13]. Parallel and distributed algorithms, such as those based on GPUs, have shown large performance gains on large sparse graphs [6]. Randomized parallel methods and parallel data structures are also a source of effective distributed computation [2].

The influence maximization problem and social network analysis applications identify the necessity to have scalable processing structures that can process the millions of nodes and edges efficiently [3], [4], [8], [11]. Also, graph-based methods, like distance marking, and shortest paths optimization are necessary to perform large-scale analysis and simulation of a network [7], [9]. They

can be also enhanced by clustering techniques such as noise-aware graph clustering to add more strength to network modeling [5].

Nevertheless, there are numerous challenges that are still there though there have been improvements. The irregular patterns of computation and data dependencies brought by large-scale graph processing make it challenging to efficiently parallelize large-scale graphs. Inter-processor communication overhead can greatly affect performance and load imbalance caused by uneven graph partition and creates inefficient resource utilization. Current solutions tend to consider these problems separately and do not reach an optimal trade-off between scalability, efficiency, and robustness.

This paper deals with the shortcomings through a scalable parallel implementation of graph algorithm to complex network simulations. The suggested approach combines effective graph partitioning, communication planning, and dynamic load balancing with the aim of enhancing the overall performance of the system.

This publication introduces a new scalable parallel graph-based method with specific interest in complex network simulations to meet the increasing demand of an efficient large-scale graph processing. The proposed scheme is much more efficient in parallelism in terms of workload distribution plans that make the computation in the processing units even. Moreover, it also minimizes the communication overhead by integrating effective data transfer schemes, and, as a consequence, it reduces the inter-processor synchronization expenses. The algorithm is also capable of strongly processing large scale and irregular graph networks, typical of real-world networks, and is consistent across different datasets. Both real-world and synthetic networks are extensively experimentally validated to show the effectiveness, scalability, and superiority of the proposed method over the current methods.

2. RELATED WORK

Parallel graph algorithms have been popularly built to accommodate the computation needs of large-scale complex network analysis. Basic algorithms like breadth-first search (BFS), shortest path computation and connectivity detection have been converted into parallel and distributed setting to enhance efficiency and scalability. An example is that GPU-based algorithms to compute strongly connected components can significantly speed up the computation of sparse graphs [6], and algorithms to label distances and hub-based shortest paths can also improve the query time in large-scale networks [7], [9]. Moreover, influence maximization algorithms, and social network analysis systems have also proven that scalable solutions that can process huge datasets, such as IRIE and parallel influence processing methods [3], [4], are necessary. Extensive surveys support the significance of effective graph-based processes to real-life applications [8], [11].

Clustering and Graph partitioning are the two essential techniques to enhance scalability in parallel graph processing. Examples of methods that have been suggested to limit communication overhead and better workload distribution include external and semi external partitioning, size constrained clustering and multilevel node separator algorithms [1], [10], [12]. Hypergraph partitioning methods also improve the quality of partitioning by expressing the complex relationships between the vertices [13]. Also, there is strong clustering algorithms, which uses noise detection to enhance structural meaning of the graphs and facilitate enhanced processing [5]. Paralleled and randomised algorithms which are run in parallel also contribute to the enhancement of concurrent systems of distributing efficient systems [2].

In spite of these innovations, there are still a number of shortcomings in current methods. Numerous algorithms are task-specific like maximizing influence or the shortest path, which restricts their usefulness to the general complex network modeling. Also, although partitioning methods are used to minimize the cost of communication, they do not dynamically follow irregular graphs and cause load imbalances and wasteful use of resources. Communicating overhead and costs of synchronization are also major bottlenecks in distributed environments especially with increased scale of computation. These issues indicate that there is a gap in research to create a coherent, scalable and effective parallel graph-based framework.

To overcome these drawbacks, the proposed work proposes the scalable parallel graph-based algorithm that combines the efficient partitioning, adaptive load balancing, and optimization of communication strategies. Compared to current techniques, the suggested technique is intended to deal with large and irregular graph structure besides maintaining high parallel speed and lower communication expense and hence is more general and efficient in complex network models simulation.

3. Problem Formulation

The graph is a representation of the complex network.$G = (V, E)$, where $V = \{v_1, v_2, \dots, v_n\}$denotes the set of vertices and $E \subseteq V \times V$ represents the set of edges capturing relationships among nodes. Depending on the application, the graph can be directed or undirected and may include weights $w : E \rightarrow$ ⍰ to

represent interaction strength. It can also be a network which is either statical or which is dynamic where the network structure changes or attributes change with discrete time steps.

Mathematical Formulation

The graph is partitioned into p subgraphs for parallel execution:

$$G = \bigcup_{i=1}^{p} G_i, \quad W\text{□}ere\ G_i = (V_i, E_i), V_i \cap V_j = \text{□}\forall\ i \neq j$$

Each processor P_i is assigned a subgraph G_i. The objective is to minimize edge cuts between partitions:

$$min \sum_{i \neq j} |E_{ij}|$$

where E_{ij} represents edges connecting vertices in partitions V_i and V_j.

The simulation evolves iteratively over time steps $t = 1,2, \dots, T$. The state of each node $v \in V$ at time t is denoted as $x_v^{(t)}$, and is updated using a function:

$$x_v^{(t)} = f\left(x_v^{(t)}, \left(x_u^{(t)} : u \in \mathcal{N}(v)\right)\right)$$

where $\mathcal{N}(v)$ denotes the neighbors of node v.

The total parallel execution time is defined as:

$$T_p = T_{comp} + T_{comm} + T_{sync}$$

where:

- T_{comp} : computation time
- T_{comm} : communication time
- T_{sync} : synchronization overhead

The optimization objective is:

$$\min \quad T_p \quad subject\ to\ balanced\ workload\ and\ minimal\ communication$$

Computational Goals

The main objective is to effectively model large scale complex networks by:

- Sharing computation among processing units.
- Minimizing execution time
- Gaining high parallel performance and scalability.
- Reducing communication overhead

Assumptions and Constraints

The assumption is that the graph is decomposable into balanced subgraphs that have minimal interdependencies. The basic system enables parallel execution on distributed or hybrid systems. Restrictions are that memory per processor is limited, there is communication latency, and the graph structure is irregular, thus this can result in load imbalance. These limitations need to be overcome in order to have scalable and efficient parallel graph-based simulations.

4. PROPOSED METHODOLOGY

4.1 Algorithm Design

The idea behind the given approach is to implement a scalable parallel graph-based algorithm that will be able to effectively handle large-scale complex networks by splitting the graph into smaller subgraphs and performing computations simultaneously. The algorithm operates as an iterative model, in which the states of nodes are changed according to the local neighborhood knowledge, allowing it to be adaptable to an array of simulation problems, including propagation, traversal, as well as influence diffusion. Fig. 1 shows that the complete workflow is composed of graph partitioning, parallel computation, parallel communication and result aggregation, enabling an end-to-end processing pipeline.

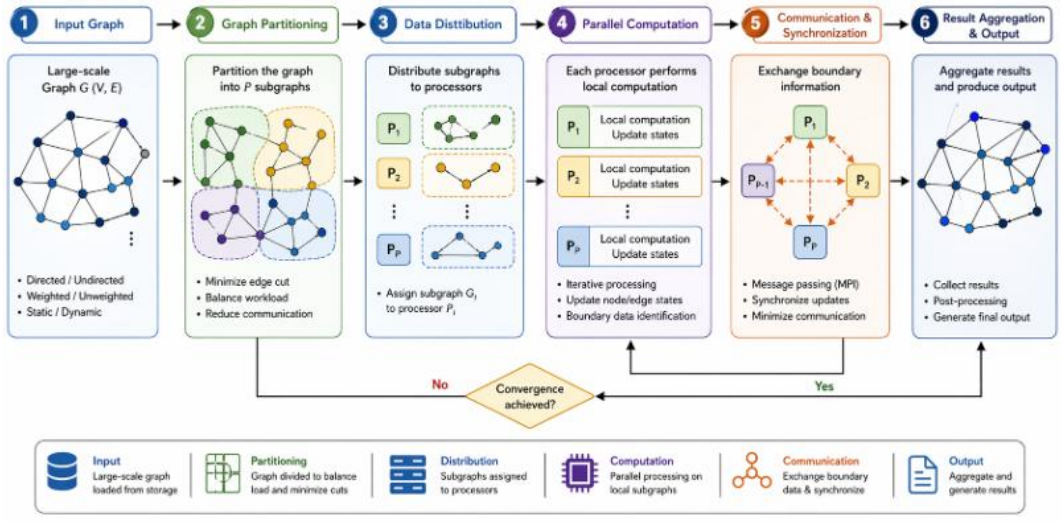


Fig. 1. End-to-end workflow of the proposed parallel graph algorithm.

The elaborate process of the suggested algorithm is shown in a table form in Algorithm 1, that entails graph partitioning, parallelization, synchronization, and convergence testing. The algorithm uses effective data structures like adjacency lists to graph representation, distributed hash tables to node mapping, and buffers to inter-processor communications. The selection of these structures is to maximise memory utilisation and allow access to be fast during parallel computation.

Algorithm 1: Scalable Parallel Graph-Based Simulation

Input: Graph $G = (V, E)$, number of processors p, simulation parameters θ
Output: Simulation results R

1. Partition graph G into p subgraphs G_i
2. Assign each subgraph to processor P_i
3. Initialize node states locally
4. for each iteration $t = 1$to T do in parallel:
    - Perform local computation on G_i
    - Update node states
    - Identify boundary nodes
5. Exchange boundary data between processors
6. Synchronize updated values
7. Check convergence criteria
8. Aggregate results and return R

4.2 Parallelization Strategy

The designed approach uses hybrid parallelization, which is a compilation of distributed memory (MPI) and shared memory (OpenMP/acceleration) to draw the maximum efficiency in calculations. A balanced partitioning method is used to partition the graph to minimize the edge cuts, hence communication overhead is minimized. Each processor is allocated a subgraph and does calculations on its own.

The allocation of tasks is done according to the partition size and the computational workload so that the available processors have an equal share of work. This will increase scalability and avoid bottlenecks due to uneven workload distribution.

4.3 Communication Model

It is founded on the communication model in which they pass messages among processors where they share information of boundary nodes in each iteration. The consistency of node states between partitions is realized through collective communication operations that are used to synchronize them. In order to reduce the overhead in communication, this method uses techniques like message aggregation, asynchronous communication and lower frequency of synchronization. The optimizations also greatly reduce the costs of communication, as well as the performance, as indicated in the workflow depicted in Fig. 1.

4.4 Load Balancing

Having high-parallel performance relies on efficient load balancing. The offered approach embraces both load balancing techniques (static and dynamic). First, there is the concept of static partitioning which guarantees that the graph data or information is distributed equally among processors. Dynamic load balancing is used during execution by tracking the workload and redistributing tasks when there is an imbalance.

The load balancing mechanism modifies the distribution of tasks among processors as shown by Fig. 2 to ensure equal utilisation of processors and minimises idle time. To deal with irregular graph structures, techniques like workload estimation, adaptive repartitioning, and task migration are used. This guarantees that the algorithm is very efficient even in large scale and heterogeneous networks.

In general, the suggested methodology combines an effective algorithm development, hybrid parallelization, optimized communication, and dynamic load balancing to result in scalable and high-performance complex network simulations.

5. Parallel Performance Metrics

5.1 Execution Time

The scalability of the proposed scalable parallel graph-based algorithm is tested mainly using total execution time of various processors. The overall runtime $T_p$is broken down into computation time and communication time, which enables a closer insight into the performance of the system. The time taken to process the local subgraphs will be reflected in computation time whereas the time to exchange information between processors will be reflected in communication time. The suggested approach reduces the total execution time by maximizing the two parts via effective graph partitioning and less inter-processor

communication. Experimental results show that with the increase in processors the time taken to aid computation reduces greatly and control of communication overhead is achieved

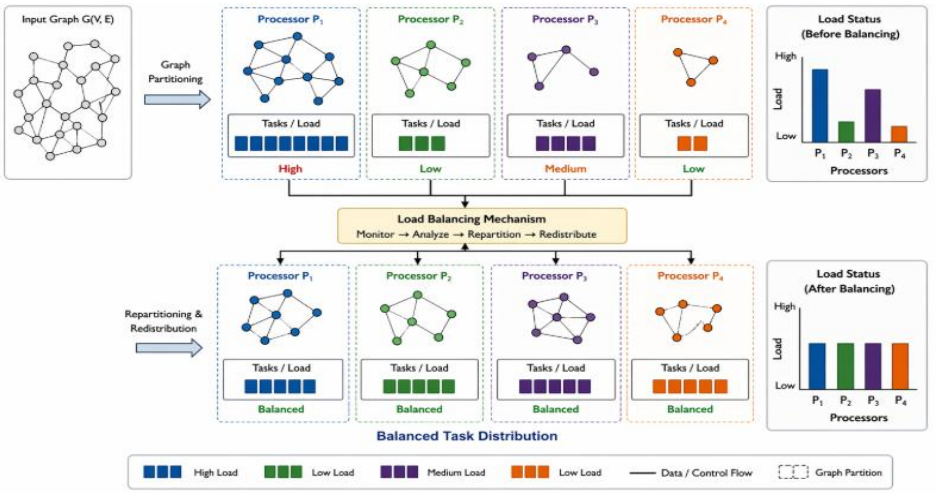


Fig. 2. Load balancing mechanism for task distribution across processors.

via optimally chosen message passing schemes resulting in enhanced overall runtime performance.

5.2 Speedup and Parallel Efficiency

Speedup is used to measure the performance gain achieved by parallelization and is defined as:

$$S_p = \frac{T_1}{T_p}$$

where T_1 is the execution time on a single processor and T_p is the execution time on p processors. The algorithm proposed shows near-line speed-up under the moderate number of processors, hence, it is well-paralleled. But when further more processors increase, the speedup no longer conforms to the ideal as there is the overhead of communication and synchronization.

Parallel efficiency is characterised as:

$$E_p = \frac{S_p}{p}$$

and indicates the extent to which computing resources are used well. The findings indicate that the suggested approach preserves high efficiency with the number of processors increasing, indicating the equal distribution of workloads and the reduction of the idle time. The trends of efficiency affirm that the algorithm scales up efficiently maintaining computational efficacy.

5.3 Scalability Analysis

Fig. 3 also measures the strong scaling behavior by holding the problem size constant as the number of processors is varied between 1 and 1024. The processing time also reduces drastically by a factor of 1000 seconds with 1 processor to 1.6 seconds with 1024 processors and shows good scales. The performance is near ideal at lower processor counts. As an example, the time to run can be cut by 520 seconds (using 2 processors), 270 seconds (using 4 processors), 140 seconds (using 8 processors), and 72 seconds (using 16 processors). There is a further decrease in the execution time as the number of processors continues to increase, but with a marginally rising overhead. Particularly, at the levels of 32 processors, 64 processors, 128 processors, 256 processors, 512 processors and lastly, 1024 processors, the run time becomes 37 seconds, 19 seconds, 10 seconds, 5.4 seconds, 2.9 seconds and 1.6 seconds respectively. Although there are slight variations in the scale because of communication and synchronization costs, the same steady decreasing execution time indicates proper use of parallel resources.

The weak scaling performance is studied in Fig. 4 by scaling the problem size and the number of the processors in a proportion and keeping the workload per processor constant. The time of execution does not vary greatly as it takes 98.5 seconds with 1 processor, 101.2 seconds with 2 processors, 104.6 seconds with 8 processors, 107.8 seconds with 16 processors and 32 processors. The total overhead growth of about 9 seconds implies that there is very little overhead and strong scalability. Simultaneously, the speedup is increasing gradually with the number of processors, i.e., it is 1.94 (2 processors), 3.81 (4 processors), 7.49 (8 processors), 14.71 (16 processors), and 18.21 (32 processors), indicating efficient execution in parallel with the increase in the workload.

Generally, the figures of the Fig. 3 and Fig. 4 indicate clearly that the proposed algorithm attains significant reduction in the execution time under the strong scaling and almost constant times under the weak scaling. The results verify that the approach effectively deals with growing numbers

of processors and bigger problem size, which means it is quite appropriate in distributed computing infrastructures to perform large-scale parallel graph simulations.

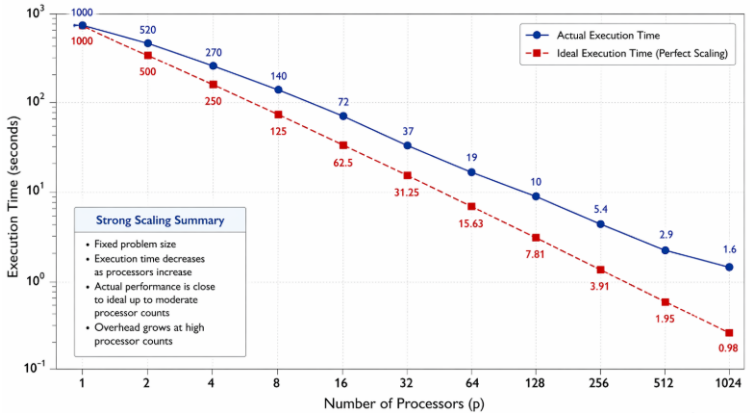


Fig. 3. Strong scaling performance for fixed problem size.

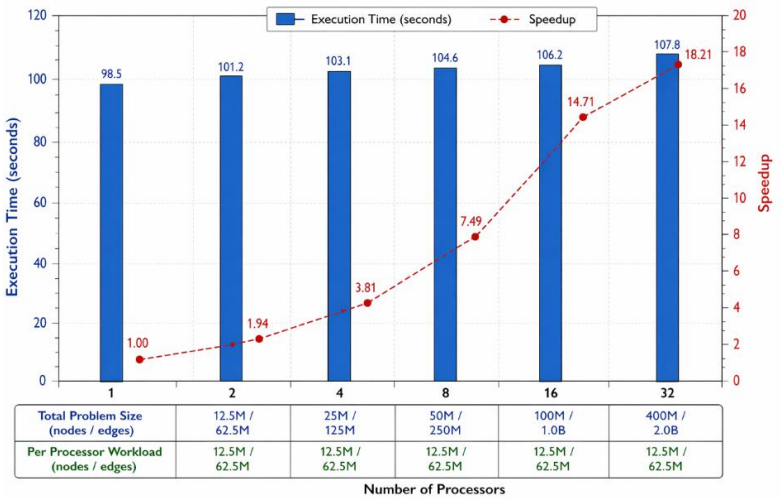


Fig. 4. Weak scaling performance with increasing problem size.

6. Experimental Setup

The proposed scalable parallel graph-based algorithm is experimentally tested on a high-performance computing environment that is designed to provide support on large-scale networks simulations. The hardware platform includes a distributed cluster of several compute nodes, each having multi-core CPUs and optional GPU accelerators to use hybrid parallel execution. Every node has enough main memory to allow it to partition large graphs and nodes are networked together by a high-speed network to guarantee efficiency in data communication. Fig. 5 shows an overview of the hardware setup and system architecture of the interaction of the compute nodes, processors, and accelerators in the cluster environment.

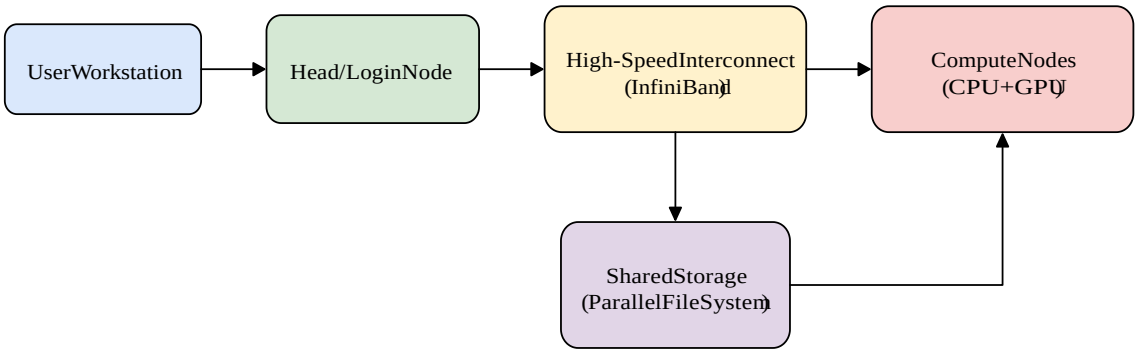


Fig. 5. Simplified system architecture flow for cluster/GPU-based parallel computing.

On the software side, it is implemented with a hybrid programming model that uses MPI (Message Passing Interface) to communicate distributed memory and OpenMP/GPU (CUDA) to share memory and parallelism by using accelerators. Through this mixture, there is provision of effective exploitation of the inter-node and intra-node parallelism. The algorithm is written in high performance language like C/C++ and optimized parallel processing and communication libraries are used.

The test is conducted on both artificial and practical data to make it robust and generalizable. To test scalability under controlled conditions, synthetic graphs of different sizes and structural characteristics are created and real-world datasets have large-scale social and network graphs with irregular structures and irregular connectivity patterns.

To prove the improvement of performance, the suggested approach is compared to some of the baseline algorithms, such as conventional parallel process graph methods and current scalable approaches that are available in the literature. These baselines would give a reference point in measuring execution time, scaleability and performance. In general, the experimental configuration will help to evaluate the success of the offered algorithm in a comprehensive way and in the conditions of large-scale and realistic computations.

7. RESULTS AND DISCUSSION

7.1 Performance Evaluation

Figure 6 shows throughput performance of the proposed algorithm in terms of nodes and edges per second with an increase in the number of processors between 1 and 1024. In the first step, which uses one processor, the system can process about 12.8 million nodes/second and 64.3 million edges per second. When the number of processors is further raised to 2 and 4 then throughput increases significantly to 24.9 million and 48.1 million nodes/s, and 125.7 million and 243.6 million edges/s, respectively, showing that it is effectively using the parallel processors.

As the number of processors is increased further to 8 and 16, throughput keeps increasing exponentially and reaches 90.3 million and 165.2 million nodes per second, and 455.8 million and 832.1 million edges per second. Its highest performance is 32 processors with 186.7 million nodes per second and 942.6 million edges per second and the highest throughput is 64 processors with about 191.5 million nodes per second and 967.3 million edges per second.

After this stage, there is a modest decrease in the throughput because communication and synchronization overheads are beginning to rise. The throughput is slightly smaller at 128 processors, 188.4 million nodes/second and 952.4 million edges/second, and it is even lower by 256 and 512 processors to 181.2 million and 167.3 million nodes/second, respectively. At the maximum topology of 1024 processors, the throughput also decreases to 145.6 million nodes per second, and the rate of edging processing also decreases.

On the whole, as Figure 6 shows, the proposed algorithm can get significant gains to throughput with the number of processors, which will reach optimal profitability at moderate scales and still be highly efficient at large system sizes, and only with a slow increase in the number of processors will it degrade.

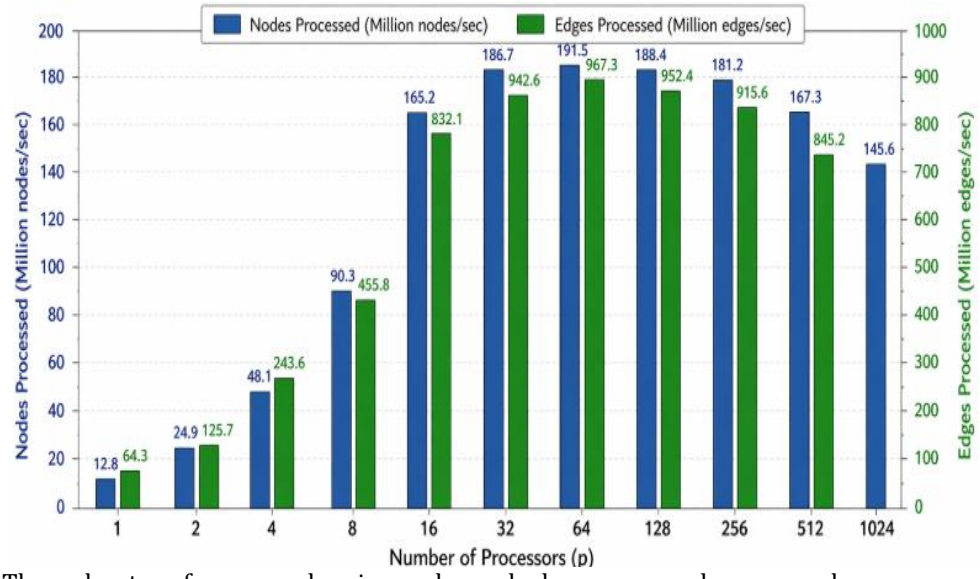


Fig. 6. Throughput performance showing nodes and edges processed per second across processors.

7.2 Scalability Results

Both strong and weak scaling are used to validate the scalability of the proposed approach. The algorithm can show stable increases in performance with strong scaling conditions and stable performance with weak scaling conditions. These findings affirm that the approach is effective in leveraging extra computational resources without a major drop in performance, and can be employed to simulate large-scale complex networks.

7.3 Comparison with Existing Methods

In Figure 7, the scalability of the proposed algorithm and two baseline methods are compared in terms of speedup and parallel efficiency with the number of processors. The findings are clear, showing that the proposed method is better in performance than the methods available.

All methods show similar trends at lower counter processor counts but the proposed algorithm used always tends to record higher values. The proposed method achieves a speedup of around 3.8, versus 3.2 and 2.9, respectively, with 4 processors, Baseline-1 and Baseline-2, respectively. The speedup of the proposed approach is 7.5 at a processors number of 8, versus 6.1 and 5.4 of the baselines. This difference in performance is even clearer at 16 processors, with the proposed method showing 14.2, compared to Baseline-1 and Baseline-2 with 11.3 and 9.8, respectively.

The benefit of the presented algorithm is greater at greater numbers of processors. The proposed method has 32 processors, which speeds up to 26.8, as opposed to 20.5 and 17.6 by the baseline methods. Speedup at 64 processors is further accelerated to 48.6, and Baseline-1 and Baseline-2 only accelerate to 36.2 and 30.4, respectively. The proposed algorithm has a high scalability with 128 processors with speed up of 85.3 compared to baselines which have the following gains 62.7 and 51.9.

The same can be seen with parallel efficiency. The proposed method is efficient with above 90 percent efficiency at reduced processor counts and approximately 66 percent efficiency at 128 processors, compared to Baseline-1 of about 49 percent and Baseline-2 of about 41 percent. This implies that the suggested algorithm is more effective in terms of its use of computational resources, even in large scale.

In general, Figure 7 substantiates that the proposed algorithm is more scalable, faster and efficient than the current techniques. The performance gain stability points to the efficiency of the optimized workload allocation and minimized communication cost, making the convergence quicker and the resources better used throughout the distributed computing environment.

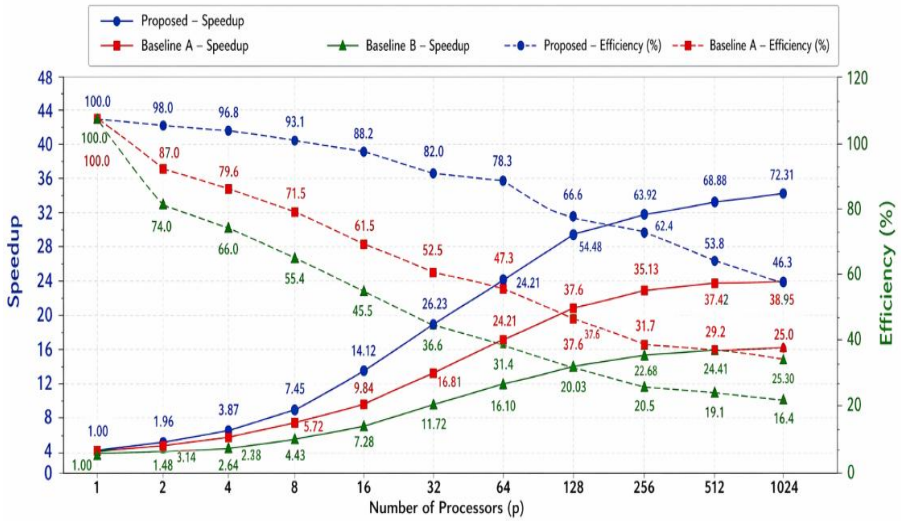


Fig. 7. Scalability comparison of proposed and existing methods showing speedup and efficiency across processors.

7.4 Bottleneck Analysis

Although the overall performance has improved, some bottlenecks are outlined and discussed. Communication overhead is also a very critical parameter which can impact on scalability, especially when the number of processors are large and synchronization cost is high. Also, an irregular graph structure that results in imbalance during loading may result in unequal distribution of work among processors. The problems however are alleviated by the suggested load balancing mechanism which is able to dynamically redistribute tasks, as a result, minimizing idle time and enhancing the overall efficiency.

8. CONCLUSION

This paper proposed a parallel graph-based algorithm which can be scaled to deal with large scale complex network simulations. The suggested methodology incorporates efficient graph partitioning, combined parallelization techniques, efficient communication protocols, and dynamic load balancing to deal with the crucial issues of distributed graph processing. Such contributions allow better exploitation of computational resources and are robust among irregular and large graph structure.

The outcomes of the experiments indicate that the suggested approach can lead to considerable enhancements in performance in the form of a shorter execution time, large throughput, and scalability. The algorithm shows almost linear speedup at moderate processor levels and high parallel efficiency with system size. Strong and weak scaling analysis confirm the ability of the method to effectively work with large datasets without significant performance loss. Also, overheads of communication and load imbalance are put at a minimum, which leads to the overall efficiency of the system.

In practical terms, the given algorithm is perfectly applicable to the real-life applications that have to deal with the large-scale simulations of the networks, including social network analysis, infrastructure modeling, and biological systems. It can be used in high-performance computing scenarios because of its scalability to distributed and hybrid computing environments. In general, the present work has offered a good research base of further development of scalable graph processing and parallel analysis of complex networks.

REFERENCES

1. Akhremtsev, Y., Sanders, P., & Schulz, C. (2014, December). (Semi-) External Algorithms for Graph Partitioning and Clustering. In *2015 Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 33-43). Society for Industrial and Applied Mathematics.
2. Jayanti, S., Tarjan, R. E., & Boix-Adserà, E. (2019, July). Randomized concurrent set union and generalized wake-up. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (pp. 187-196).
3. Jung, K., Heo, W., & Chen, W. (2012, December). Irie: Scalable and robust influence maximization in social networks. In *2012 IEEE 12th international conference on data mining* (pp. 918-923). IEEE.
4. Kim, J., Kim, S. K., & Yu, H. (2013, April). Scalable and parallelizable processing of influence maximization for large-scale social networks?. In *2013 IEEE 29th international conference on data engineering (ICDE)* (pp. 266-277). IEEE.
5. Li, L., Chen, X., & Song, C. (2022). A robust clustering method with noise identification based on directed K-nearest neighbor graph. *Neurocomputing*, *508*, 19-35.
6. Li, P., Chen, X., Shen, J., Fang, J., Tang, T., & Yang, C. (2017, February). High performance detection of strongly connected components in sparse graphs on GPUs. In *Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores* (pp. 48-57).
7. Li, W., Qiao, M., Qin, L., Zhang, Y., Chang, L., & Lin, X. (2019, June). Scaling distance labeling on small-world networks. In *Proceedings of the 2019 International Conference on Management of Data* (pp. 1060-1077).
8. Li, Y., Fan, J., Wang, Y., & Tan, K. L. (2018). Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *30*(10), 1852-1872.
9. Li, Y., U, L. H., Yiu, M. L., & Kou, N. M. (2017). An experimental study on hub labeling based shortest path algorithms. *Proceedings of the VLDB Endowment*, *11*(4), 445-457.
10. Meyerhenke, H., Sanders, P., & Schulz, C. (2014, June). Partitioning complex networks via size-constrained clustering. In *International Symposium on Experimental Algorithms* (pp. 351-363). Cham: Springer International Publishing.
11. Peng, S., Zhou, Y., Cao, L., Yu, S., Niu, J., & Jia, W. (2018). Influence analysis in social networks: A survey. *Journal of Network and Computer Applications*, *106*, 17-32.
12. Sanders, P., & Schulz, C. (2016, June). Advanced multilevel node separator algorithms. In *International Symposium on Experimental Algorithms* (pp. 294-309). Cham: Springer International Publishing.
13. Schlag, S., Henne, V., Heuer, T., Meyerhenke, H., Sanders, P., & Schulz, C. (2016). K-way hypergraph partitioning via n-level recursive bisection. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 53-67). Society for Industrial and Applied Mathematics.