

New efficient fractal models for MapReduce in OpenMP parallel environment

Muslim Mohsin Khudhair^{1,2}, Furkan Rabee², Adil AL_Rammahi³

¹Department of Computer Information System, College of Computer Science and Information Technology, University of Basrah, Basrah, Iraq

²Department of Computer Science, Faculty of Computer Science and Mathematics, University of Kufa, Najaf, Iraq

³Department of Mathematical Science, Faculty of Computer Science and Mathematics, University of Kufa, Najaf, Iraq

Article Info

Article history:

Received Oct 9, 2022

Revised Nov 9, 2022

Accepted Jan 7, 2023

Keywords:

Cloud computing

Fractal

MapReduce

OpenMP

ABSTRACT

Parallel data processing is one of the specific infrastructure applications categorized as a service provided by cloud computing. In cloud computing environments, data-intensive applications increasingly use the parallel processing paradigm known as MapReduce. MapReduce is based on a strategy called "divide and conquer," which uses ordinary computers, also called "nodes," to do processing in parallel. This paper looks at how open multiprocessing (OpenMP), the best shared-memory parallel programming model for high-performance computing, can be used in the MapReduce application using proposed fractal network models. Two fractal network models are offered, and their work is compared with a well-known network model, the hypercube. The first fractal network model achieved an average speedup of 3.239 times while an efficiency ranged from 73-95%. In the second model of the network, the speedup got to 3.236 times while keeping an efficiency of 70-92%. Furthermore, the path-finding algorithm employed in the recommended fractal network models remarkably identified all paths and calculated the shortest and longest routes.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Muslim Mohsin Khudhair

Department of Computer Information System, College of Computer Science and Information Technology

University of Basrah

Basrah, Iraq

Email: muslim.khudhair@uobasrah.edu.iq or mos1970@yahoo.com

1. INTRODUCTION

The ability to offer computing resources and services on-demand has propelled the popularity of computational clouds. A collection of applications is arranged on virtual machines, which then execute on multiple server computers in data center networks (DCNs). These machines serve as the backbone for computational clouds. With the popularity of cloud computing and mobile cloud computing, DCN is the fundamental infrastructure and backbone supporting service delivery. The tree-based hierarchical structure is among the most famous architectural type [1], [2].

Big data is becoming an increasingly effective approach for a corporation to improve its value proposition or operations. Big data uses parallel computing to process large amounts of data quickly. Big data supports several specialized programming paradigms and runtime technologies [3]. Spark and Hadoop are map/reduce frameworks [4]. In GraphLab, Giraph, and GraphX, the Pregel model is implemented [5]. Storm [6] handles data streaming. Each system uses threads and simple methods to create a parallel and distributed computing environment. In contrast, the high-performance computing community developed programming

models that help build systems like the above and strike a balance between programming effort and performance.

Having cloud computing has made MapReduce easier to use. The infrastructure services (infrastructure-as-a-service) offered by cloud companies make it easy to set up large computing instances. cloud providers such as Amazon and Microsoft offer MapReduce platform services (platform-as-a-service). MapReduce clusters can now be put together at a lower cost or on a pay-per-use basis [7].

The implementation of parallel computation can be done in various ways. Generally, parallel computing on the CPU is based on MPI or open multiprocessing (OpenMP), which are for distributed memory parallelism systems and shared memory multiprocessor systems [8]. Consequently, the primary focus of this research is on using the OpenMP program to simulate Mapreduce by employing presented fractal network models to fulfil the needs for processing big data.

The remaining parts of this work are structured in the following manner. Section 2 provides a condensed explanation of hypercube and MapReduce. The basic concepts and knowledge of the parallel computing environment by OpenMP are presented in section 3. The fundamental ideas of fractals, which will serve as the foundation for the suggested architectural frameworks, are introduced in section 4. Section 5, the heart of this paper, explains the given proposal fractal models in detail. The specifics of the experiments and an analysis of the findings are presented in section 6. Section 7 wraps up the paper and discusses further research.

2. BASIC CONCEPTIONS

2.1. Hypercube

One of the best major connectivity networks in numerous topological aspects is the hypercube, often known as a n-cube. These topological qualities include a low diameter and high connectivity relative to other critical interconnection networks. A hypercube network structure improves network efficacy and stability and reduces data transfer difficulties such as complexity, speed, and size [9].

The hypercube of order n is an undirected network with 2^n nodes with labels ranging from 0 to 2^n-1 . The number of linkages that are incident on a node is represented by its degree, which is written as n. The hypercube is a building with logarithmic architecture. This is because $\log_2 N = n$ links are the maximum connections a message must go over to reach its destination in a hypercube [10]. Also, one of the things that makes the hypercube topology interesting is its diameter, which is the largest number of nodes a message must pass through to get to its mate. The Figure 1 represents the hypercube in varying dimensions [11].

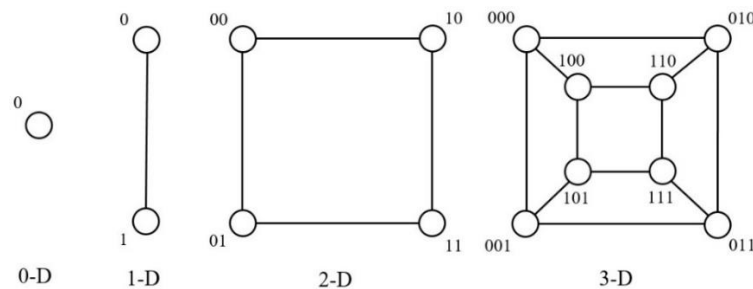


Figure 1. Hypercube with varying dimensions (n-D)

2.2. Mapreduce architecture

Google released MapReduce in 2004 to enable fast, fault-free, scalable, and load-balanced parallel processing of enormous data sets. It is used in various fields, including data mining, extensive graph processing, machine learning, text processing, statistical machine translation, and others. Hadoop, built by the Apache Foundation, is one of many MapReduce implementations. Hadoop, like MapReduce, has two layers: a data storage layer called Hadoop DFS (HDFS) and a data processing layer called Hadoop MapReduce framework. HDFS, like Google's GFS, is controlled by a single master node [12].

MapReduce's master-slave concept is seen in Figure 2. The JobTracker manages tasks, progress, and resources as the central controller. TaskTrackers are slaves in this architecture. They start and monitor local mapper and reducer activities and provide JobTracker with updates and reports. The input data is automatically split into smaller parts and processed in parallel by a mapper. After mapping the intermediate data, the reduction stage produces the query's final result [13].

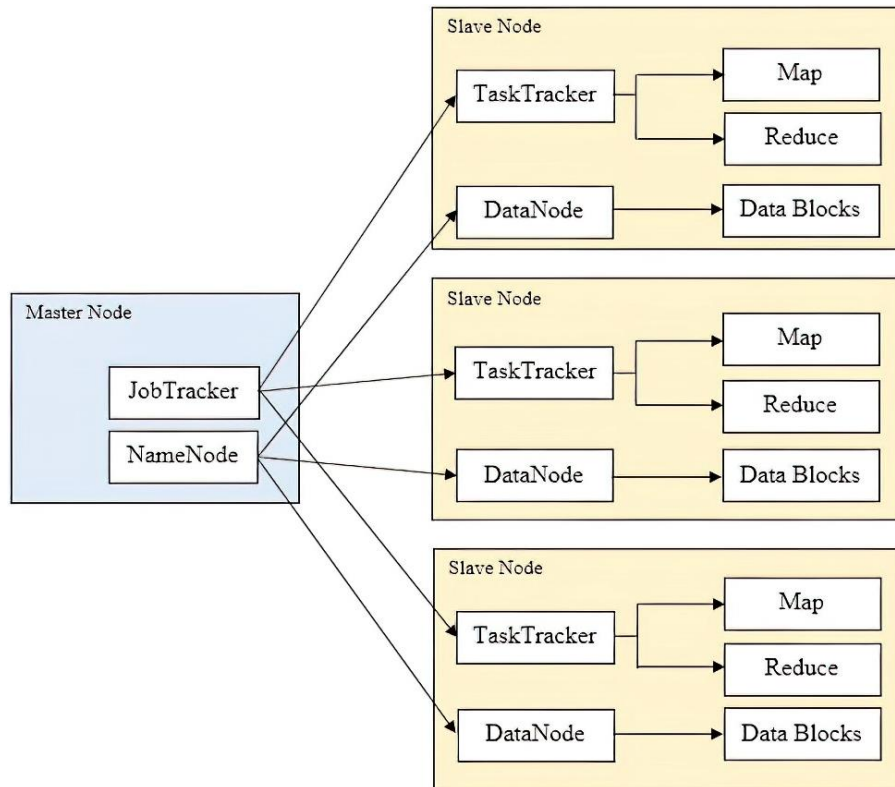


Figure 2. The basic architecture of MapReduce

2.2.1. MapReduce framework

There are two parts to the MapReduce model: map and reduce, which work on (key, value) pairs. The following describes the process flow of the execution of MapReduce [13], [14]:

- The input reader splits the input into several smaller components (chunks). After that, a map function is given responsibility for these blocks.
- In the map function, each element is partitioned into tuples referred to as (key, value) pairs.
- Within the shuffle function, the outputs of the mappers are moved to reducer nodes by the values of their respective keys.
- The reduce function takes the intermediate tuples and merges them into a smaller collection of tuples, which are then sent to the output.
- The job of the output writer is to ensure that the output is properly saved to storage, which may take the form of a database or a file system.

3. OPEN MULTIPROCESSING

Symmetric multiprocessor platforms (SMPs) with numerous CPU cores are best for fine-grained parallel computation using OpenMP. OpenMP-based computing reduces bandwidth and latency [15]. Compiler directives enable loop-based parallelization, tasking, work-sharing, and synchronization in C/C++ and Fortran. Because it's easy to run sequential applications in parallel, the OpenMP architecture is more enticing to programmers [16]. OpenMP's primary parallelization technique is known as multithreading. As shown in Figure 3, a single master thread separates multiple slave threads. In the serial context, the master thread executes instructions sequentially. The slave threads in the parallel environment execute instructions simultaneously and independently [17].

Because OpenMP uses a shared memory model, every thread can access the global memory by default. Slave threads can communicate by reading and writing to the global memory. When they update global or shared memory simultaneously, it can lead to a race condition that changes depending on how the threads are scheduled [18]. When two or more threads access the same memory without the necessary synchronization, this is called a "data race condition". Additionally, OpenMP enables the parallelization of parallel areas, allowing parallel loops to be nested inside parallel loops. When this happens, the slave threads that were made spawn more threads to make the team [19].

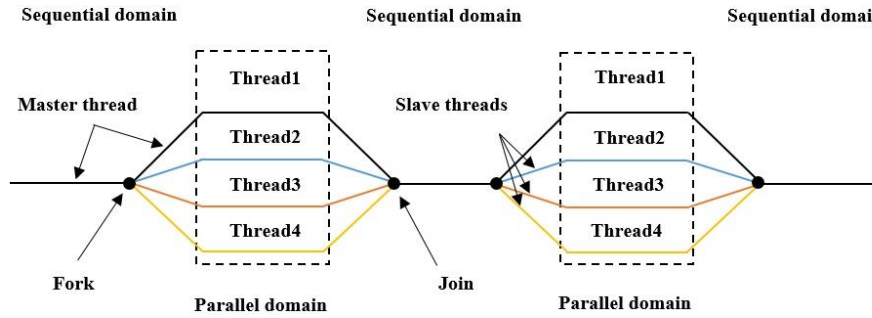


Figure 3. A fork-and-join approach of OpenMP parallelism

3.1. Scheduling methods

Scheduling work becomes the most important part of parallelization. OpenMP gives you several ways to set up a schedule, which are [17], [20]: i) the static schedule works best if all the iterations take the same time to compute; ii) dynamic scheduling gives each thread a small amount of work; after it finishes, it gets more. It improves load-balancing among OpenMP threads when a loop's iterations have uneven demand; iii) the guided schedule sets the chunk size based on the number of unallocated iterations. So, the first sections of the threads are greater. Chunk size reduces as iterations increase; iv) the auto schedule lets the compiler and the runtime decide how to schedule things. The behaviour of the automatic schedule will differ depending on the implementation-specific; and v) the runtime schedule lets a program wait until runtime to decide which OpenMP schedule to use. Schedule and chunk-size options can be chosen at runtime.

3.2. Speedup and efficiency in parallel computing

Speedup and parallel efficiency measure how well a parallel algorithm works on a parallel architecture. The equations listed are utilized in the process of determining them [19], [21]:

$$S_{Parallel} = \frac{T_{Sequential}}{T_{Parallel}} \tag{1}$$

$$E_{Parallel} = \frac{S_{Parallel}}{N_{Parallel}} \tag{2}$$

where S (*parallel*) is the parallel speedup, T (*sequential*) is the time it took for the sequential program to run, T (*parallel*) is the time it took for the parallel program to run, and E (*parallel*) is the efficiency of parallel processing, and N (*parallel*) is the number of processors used for parallel processing.

4. THE FUNDAMENTALS OF FRACTALS AND THEIR MEASUREMENTS

Fractals were first employed to describe nature's uneven geometry, such as tree branches, mountain surfaces, and shoreline features [22]. Forms have whole-number topological dimensions in euclidean geometry. A line has one dimension, a square two, and a cube three. Fractal objects need further explanation in this definition. Every natural, artificial, and random fractal has an (often non-integer) dimension that can be used to measure or quantify its complexity about its environment [23], [24]. The formula for finding the similarity dimension, D_s , for self-similar fractals with N copies that are all scaled by the same factor r , is as (3):

$$D_s = \frac{\log(N)}{\log(1/r)} \tag{3}$$

Table 1 depicts the similarity dimensions of some of the most important fractals studied (in ascending order of magnitude).

Table 1. Displays the dimension of similarity for a selection of well-known fractals

Fractal name	Similarity dimension	No. of copies (N) and scaling factor (r)
Cantor set	0.63100	$r=1/n$ at the n^{th} step of construction
Straight line	1.00000	$N=1, r=1$
Koch curve	1.26186	$N=4, r=1/3$
Box fractal	1.46497	$N=5, r=1/3$
Sierpinski triangle	1.58496	$N=3, r=1/2$
Sierpinski carpet	1.89279	$N=4, r=1/4$

4.1. Sierpinski triangle

The widely recognised Sierpinski triangle (Sierpinski, 1916), seen in Figure 4, is built using an iterative process. The Sierpinski triangle is equilateral when the original configuration at initial iteration $m = 0$ is denoted by the label S_0 . In the initial generation S_1 , we choose the intersection of the three sides of the equilateral triangle (l), connect them to form four different triangles, and then eliminate the central triangle. Like the previous generation, the second generation S_2 links three midpoints of the sides ($l/2$) in each little triangle. And to be more comprehensive, if we keep repeating the technique for all new little triangles, we will eventually arrive at the basic deterministic fractal known as the Sierpinski triangle when the value of $m = \infty$. This means one of the critical characteristics of the Sierpinski triangle is its ability to contain an infinite number of triangles inside a small space S_0 [22], [25].

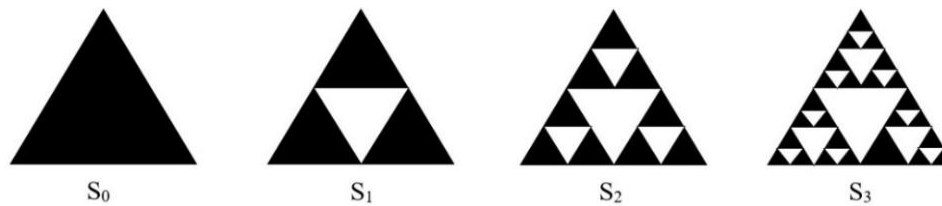


Figure 4. Sierpinski triangle with four levels

4.2. Sierpinski carpet

The Sierpinski carpet is an additional example of an ideal fractal object analogous to the Sierpinski triangle. The carpet is distinctive since its main components are square. Starting with a square fundamental particle S_0 , a bottom-up building method is described. To create a stage S_1 aggregate, first, arrange eight primary elements into a square loop. Then, leave a square hole in the center of the loop that is the same size as the primary elements. For a stage S_2 aggregate, put eight stages S_1 aggregates in a square loop and leave a hole in the middle that is the same size as a stage S_1 aggregate (see Figure 5). Since the fundamental particles have little touch with one another, the intrinsic perimeter of the Sierpinski carpet is not equal to the total of their edges, as is the case with the triangle [22], [25], [26].

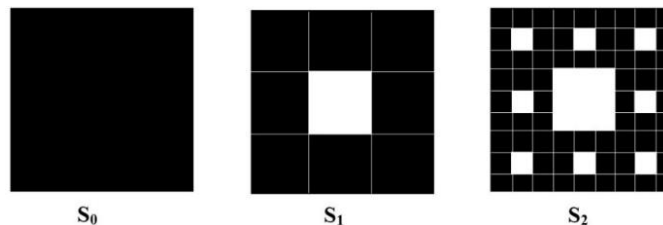


Figure 5. The first three stages of the Sierpinski carpet

4.3. The fractal models storage algorithm

A data structure can be stored in memory as a graph representation using linked lists [27], [28]. The algorithm used to define the relationships and storage of nodes across multiple diagrams is represented by as shown in Figure 6. This algorithm stores the suggested model fractal architectures in memory using a programmed implementation of those architectures. It is possible to get a linkage between the nodes, which is shown to be represented by the adjacency matrix and link lists in Figures 7-9.

4.4. The fractal models routing algorithm

Understanding how the potential routes between the nodes of the suggested model fractal architecture are crucial [29], [30]. Therefore, as illustrated in Figure 10, a routing algorithm is developed for these structures by taking advantage of the storage schemes from the previous storage algorithm. This algorithm has been programmed into the proposed model fractal architectures. Using Figure 7 as an example, which has six nodes, the following is the output of the program that was used to find the possible paths between nodes 0 and 5: ([0, 3, 1, 5], [0, 3, 5], [0, 3, 4, 2, 5], [0, 3, 4, 5], [0, 4, 2, 5], [0, 4, 3, 1, 5], [0, 4, 3, 5], and [0, 4, 5]).

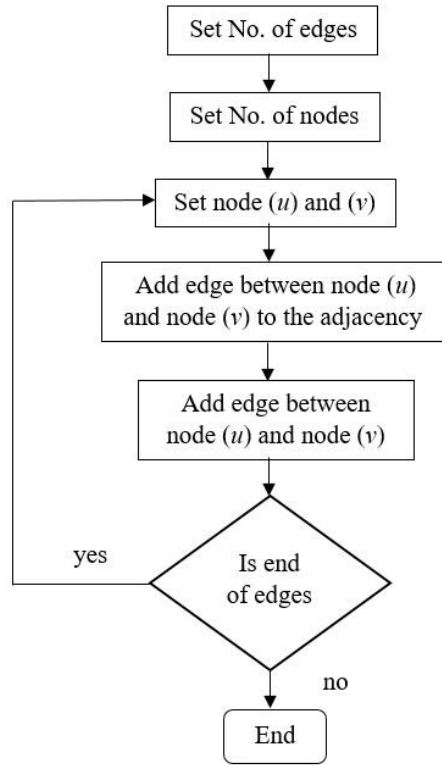


Figure 6. Storage algorithm of the model fractal architectures

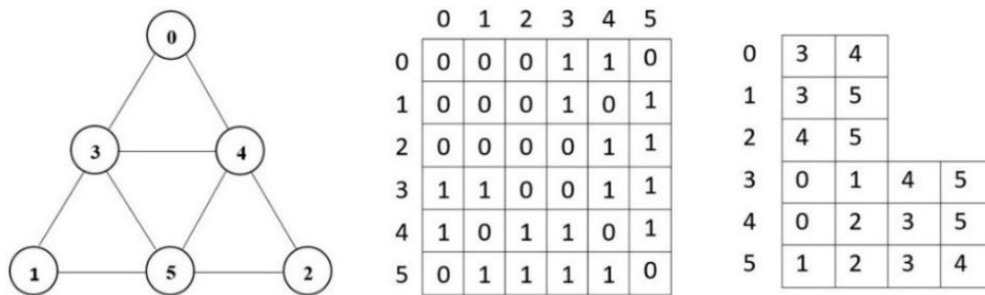


Figure 7. Storage data of the Sierpinski triangle (6 nodes)

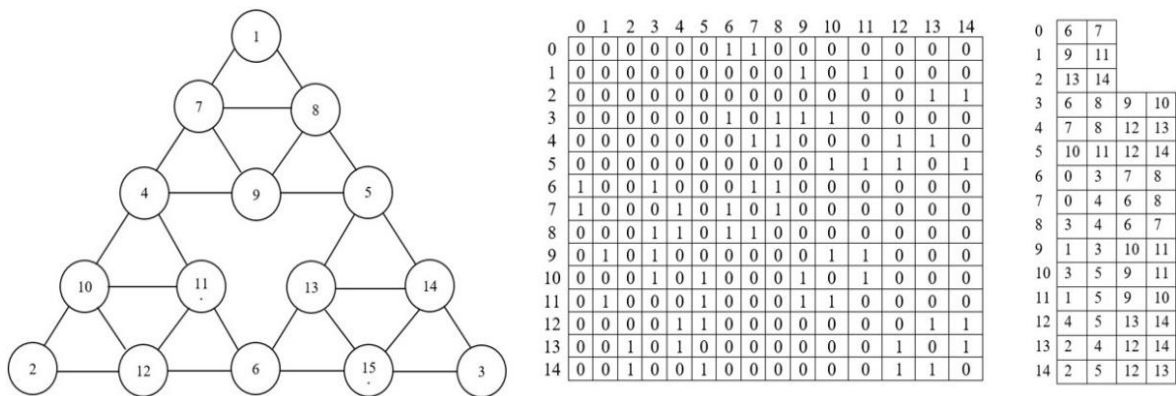


Figure 8. Storage data of the Sierpinski triangle (15 nodes)

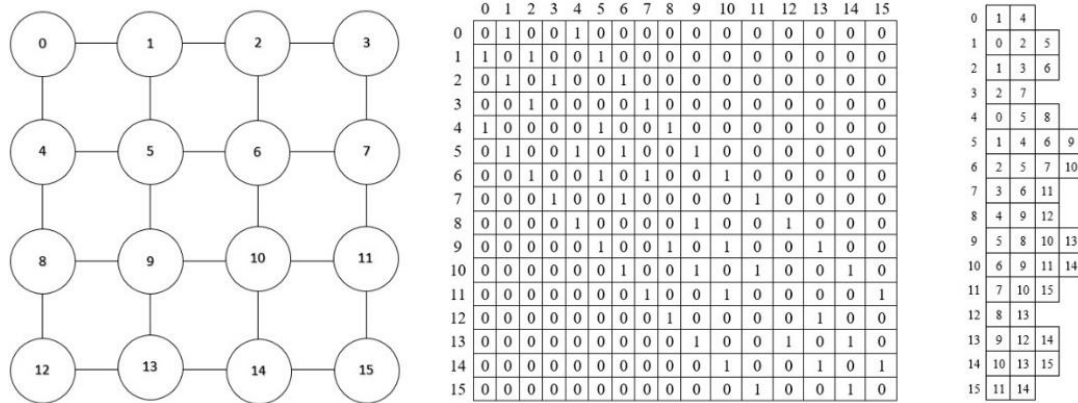


Figure 9. Storage data of the Sierpinski carpet (16 nodes)

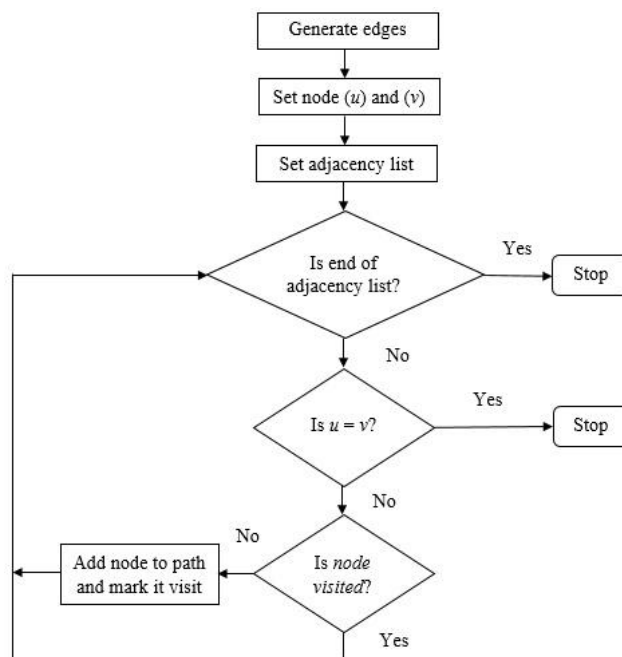


Figure 10. Routing algorithm of the model fractal architectures

5. PROPOSED FRACTAL MODELS

The prototype of the Sierpinski triangle architecture is based on the fractal recursive formula Sierpinski triangle (gasket) described in subsection 4.1. This model system positions processing elements at the graph's vertices (nodes). The word count is an example of how efficiently the model network works. Word count is a typical use of MapReduce, which reads text files and counts how many times each word appears. The overall number of lines in the text input file is counted right at the beginning of the test for every model. The file is then split into chunks, each with a certain number of text lines. In the context of OpenMP, the simulations and actualizations of the following model architectures are being carried out.

5.1. The first model fractal architecture

A detailed explanation of the procedures involved in building the model, by the following steps:

- Counting the number of lines of the entered text data file.
- Splitting the lines of the text file into equal chunks and distributing these chunks regularly to the network nodes, as in Table 2.
- In the first stage, a network is allocated so that each node represents a processing unit that performs the mapping and shuffling process.
- Under the OpenMP environment parallel processing, each node is assigned a free thread to perform the required processing (mapping and shuffling).

- The network outputs for this stage are (value, keys) stored in text files and considered inputs to the second stage, as in Figure 11.
- In the second stage, within the OpenMP environment, network nodes are allocated to perform the Reducing process in the same way as the processing nodes' data distribution (chunks) in the first stage.
- Finally, the output of this stage is the completion of the MapReduce operation to calculate the number of word repetitions for the input text file.

The algorithm of the first model fractal architecture takes an input data file (IPs) and returns an output file for repeating words (IPs). The Algorithm 1 describes the main essential steps implemented within the OpenMP environment.

Table 2. Represents a method for distributing chunks of a text input file to network processing nodes

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
chunk 1	chunk 2	chunk 3	chunk 4	chunk 5	chunk 6
chunk 7	chunk 8	chunk 9	chunk 10	chunk 11	chunk 12
chunk 13	chunk 14	chunk 15	chunk 16	chunk 17	chunk 18
..	chunk n

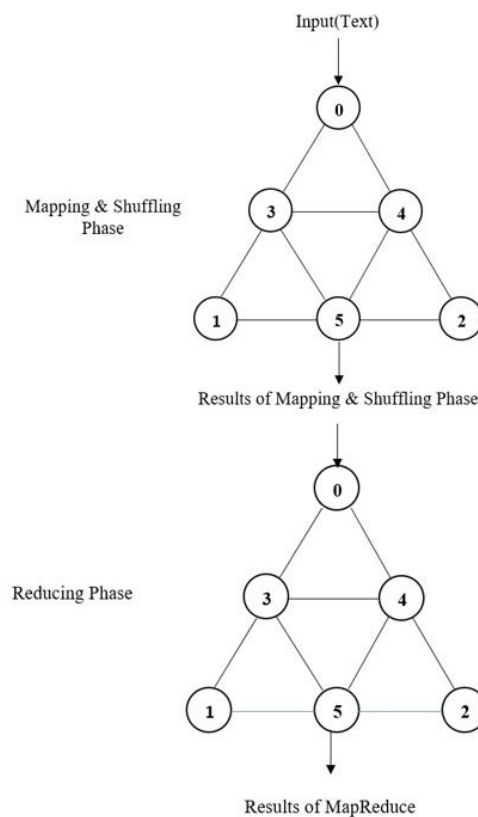


Figure 11. The first model fractal architecture for implementing MapReduce

Algorithm 1. The first model of fractal architecture

Input:

```

fin: input file of line as IPs
chunkNo: number of chunks
nodeNo: number of nodes
threadNo: number of threads

```

Output:

```

fo: file of occurrences of IPs

```

Begin

```

1: set threadNo=6
2: call omp_set_num_threads(threadNo)
3: #pragma omp parallel shared ()
4: num=call omp_get_thread_num ();
5: #pragma omp single

```



```

6:     for (index=1; index<=chunkNo; index++)
7:         Node=index % 6                                //Specify node number
8:         #pragma omp task
9:         if (Node) identical thread No. (num)
10:            call mapping (chunk_index, fo_map)
11:            call shuffling (fo_map, fo_shuffle)
12:         end if
13:         #pragma omp taskwait
14:     end for                                           //End phase mapping &
shuffling
15: #pragma omp parallel shared ()
16: #pragma omp single
17:     for (index=1; index<=chunkNo; index++)
18:         Node=index % 6
19:         #pragma omp task
20:         if (Node) identical thread No. (num)
21:            call reducing (fo_shuffle, fo_reduce)
22:         end if
23:         #pragma omp taskwait
24:     end for                                           //End phase reducing
25:     aggregating th previous output reducing to get the final output file (fo)
26: end begin

```

5.2. The second model fractal architecture

In this model, a single network model is used to process the three operations of the MapReduce during one phase, as shown in Figure 12. So that the text file data splitting and distribution are similar to the first model. The second model fractal architecture algorithm returns an output file for repeated words from an input data file (IPs). OpenMP's major phases are described in Algorithm 2.

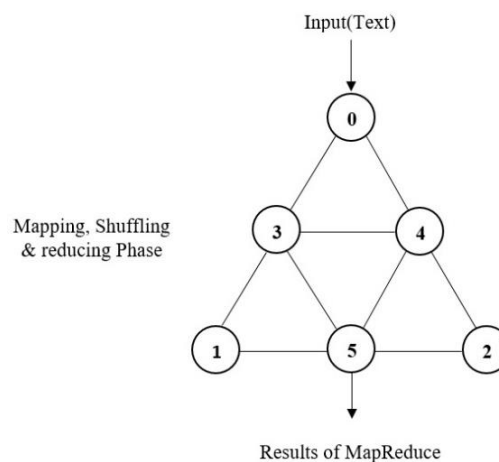


Figure 12. The second model fractal architecture for implementing MapReduce

Algorithm 2. The second model of fractal architecture

Input:

```

fin: input file of lines as IPs
chunkNo: number of chunks
nodeNo: number of nodes
threadNo: number of threads

```

Output:

```

fo: file of occurrences of IPs

```

Begin

```

1: set threadNo=6
2: call omp_set_num_threads(threadNo)
3: #pragma omp parallel shared ()
4: num=call omp_get_thread_num ();
5: #pragma omp single
6: for (index=1; index<=chunkNo; index++)
7:     Node=index % 6                                //Specify node
number
8:     #pragma omp task
9:     if (Node) identical thread No. (num)
10:        call mapping (chunk_index, fo_map)

```

```

11:         call shuffling (fo_map, fo_shuffle)
12:         call reducing (fo_shuffle, fo_reduce)
13:     end if
14:     #pragma omp taskwait
15: end for
16: aggregating the previous output reducing to get the final output file (fo)
17: end begin
    
```

6. EXPERIMENTAL FINDINGS AND ANALYSIS

In this section, we will study and discuss the outcomes of serial and parallel implementations of recommended fractal architectural models and compare them to well-known architectural structures. The experiment used a quad-core HP Laptop (60 GHz CPU, 16 GB RAM). C++ implements the experiment program. Using gcc 4.9.2 and OpenMP 5.0 on Windows, we compiled the apps. Six text files were tested (data 1, data 2, data 3, data 4, data 5, and data 6). Each file had data concerning IP addresses on lines that were (10,000; 200,000; 300,000; 400,000; 500,000; and 600,000), accordingly.

6.1. Speedup and parallel efficiency

For the first model, the runtime calculation performs for various thread counts ranging from 1 to 12 using the OpenMP explicit tasks functionality, as displayed in Figure 13. As a result, it notes that execution time dropped by an average of 69% for text data sets. The input data of this model is divided into chunks for a second test. The amount of time needed to process data sets changes depending on the size of the chunk being processed, as shown in Figure 14. As a result, it is thought to be a factor impacting network performance.

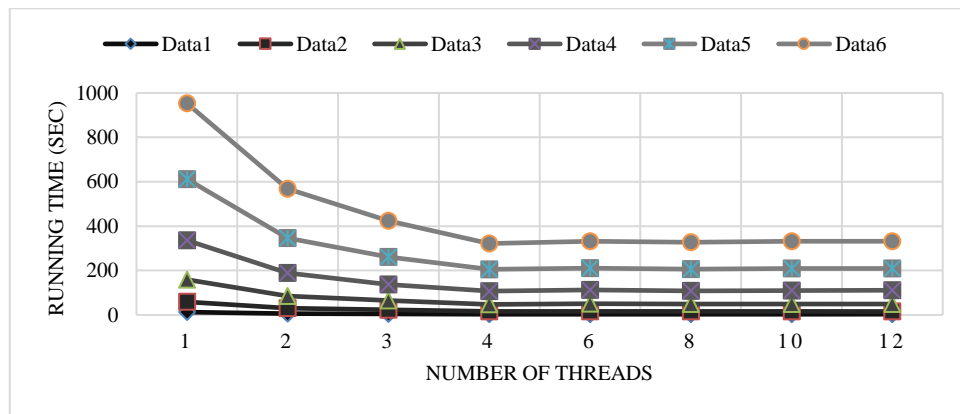


Figure 13. The running time for multiple distinct data sets with varying numbers of threads

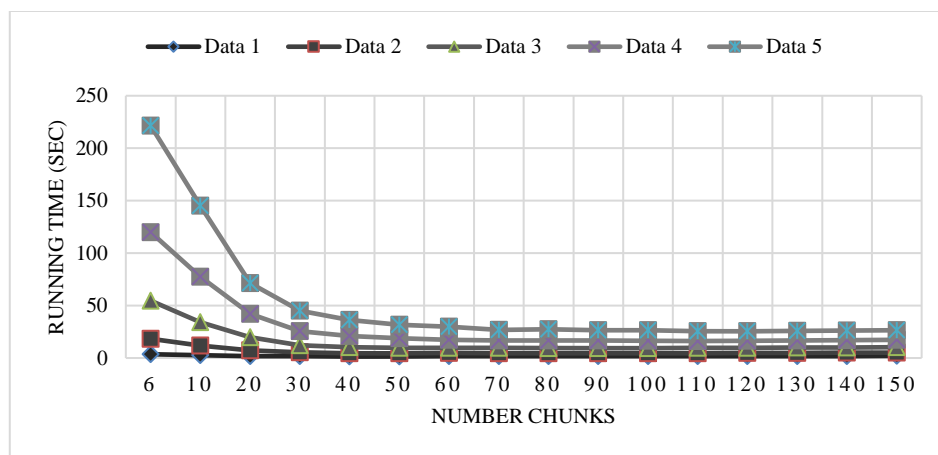


Figure 14. The running time for several data sets with various chunk sizes runs parallel

The third test of the model shown in Table 3 illustrates two measures to assess a program's effectiveness: speedup and efficiency. The graph that can be seen in Figure 15 shows the relationship between the speedup and the number of threads employed for text data sets. The performance of the suggested second model fractal architecture is assessed based on the same technique carried out on the first model. This performance is reflected by the Table 4 and Figures 16-18 which can be seen.

Table 3. Efficiency and speedup of the first model for diverse data sets at various thread counts

Thread no./Data sets		2	3	4
Data 1	Speedup	1.922	2.757	3.832
	Efficiency	0.961	0.919	0.958
Data 2	Speedup	1.947	2.537	3.711
	Efficiency	0.9735	0.846	0.928
Data 3	Speedup	1.819	2.394	3.197
	Efficiency	0.9095	0.798	0.799
Data 4	Speedup	1.707	2.326	2.945
	Efficiency	0.8535	0.775	0.736
Data 5	Speedup	1.756	2.238	2.810
	Efficiency	0.878	0.746	0.703
Data 6	Speedup	1.542	2.088	2.941
	Efficiency	0.771	0.696	0.735
Average	Speedup	1.782	2.390	3.239
	Efficiency	0.891	0.797	0.810

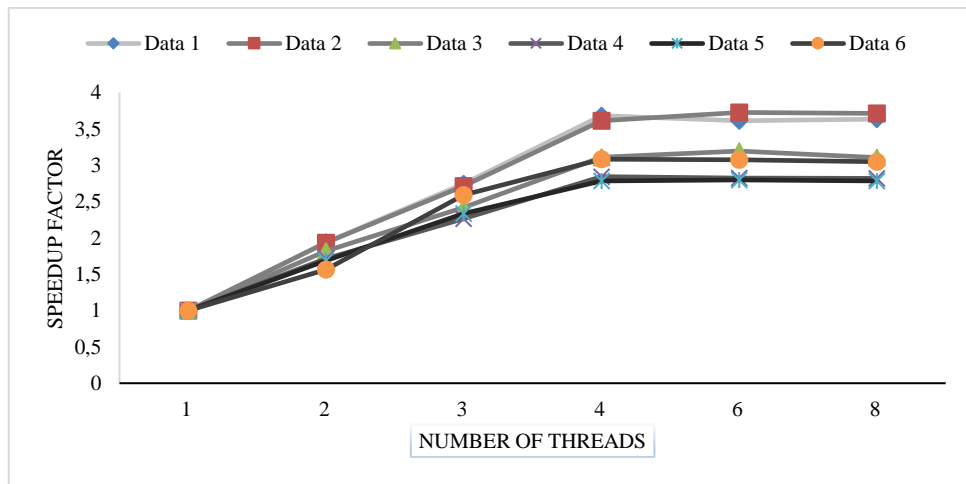


Figure 15. The running time for several data sets with different thread counts

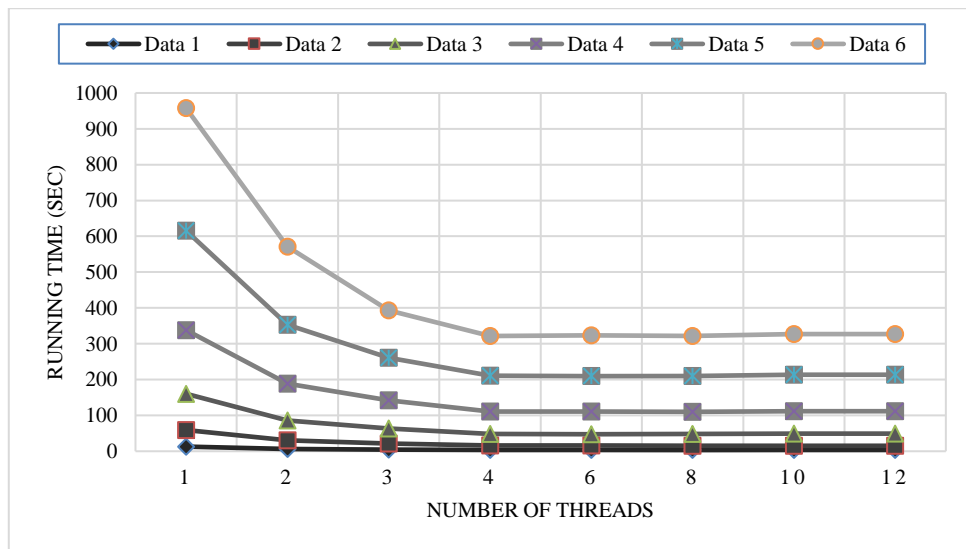


Figure 16. The running for a variety of thread counts across many data sets

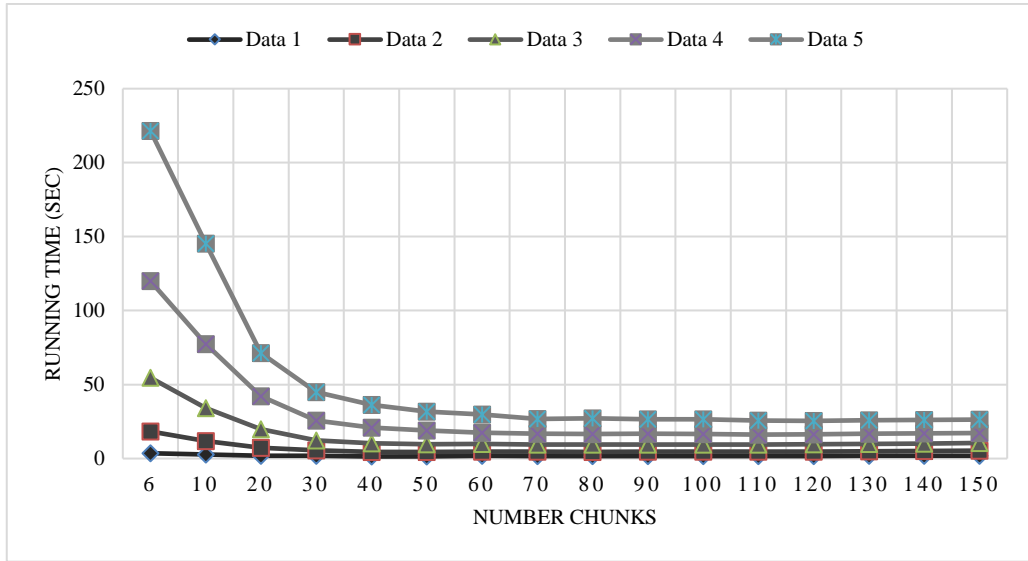


Figure 17. Duration of the first model's execution for several data sets with different thread counts

Table 4. Efficiency and speedup of the second model for diverse data sets across varying thread counts

Thread no./Data sets		2	3	4
Data 1	Speedup	1.940	2.743	3.682
	Efficiency	0.970	0.914	0.921
Data 2	Speedup	1.934	2.712	3.613
	Efficiency	0.967	0.904	0.903
Data 3	Speedup	1.820	2.419	3.110
	Efficiency	0.910	0.806	0.778
Data 4	Speedup	1.713	2.265	2.843
	Efficiency	0.857	0.755	0.711
Data 5	Speedup	1.689	2.334	2.785
	Efficiency	0.845	0.778	0.696
Data 6	Speedup	1.563	2.592	3.383
	Efficiency	0.782	0.864	0.846
Average	Speedup	1.776	2.511	3.236
	Efficiency	0.889	0.83	0.809

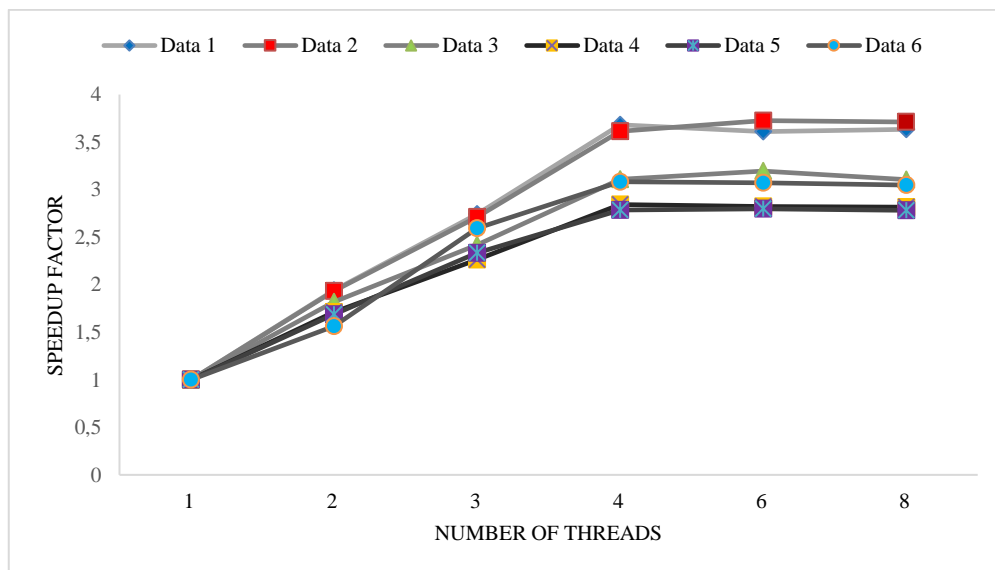


Figure 18. The running time for multiple distinct data sets with chunks of different sizes running in parallel

It uses a well-known hypercube model to compare and evaluate the performance of the newly presented fractal models, as shown in Figure 19 (described in subsection 2.1). This model evaluates using the same methods as the proposed models. First, a comparison is made between the hypercube and the first suggested model. According to Table 5 findings, the first model presented has a marginal advantage in completing the task in a shorter amount of time. When the second model is contrasted with the hypercube, the findings indicate that the hypercube is slightly preferred over the second model. This is illustrated in Table 6.

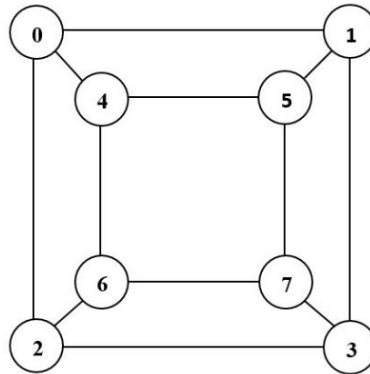


Figure 19. Traditional hypercube model architecture

Table 5. Results for data sets with different chunks of the first model and the hypercube

Data sets	Number of chunks	Running time of the first model (sec)	Running time of the hypercube (sec)
Data 3	6	31.408	36.106
	12	16.418	16.959
	18	9.939	9.999
	24	7.517	7.529
Data 4	6	60.080	64.716
	12	27.746	34.591
	18	18.485	25.981
	24	12.737	13.181
Data 5	6	98.206	96.417
	12	49.442	51.317
	18	27.645	34.895
	24	22.350	22.831

Table 6. Results for data sets with different chunks of the second model and the hypercube

Data sets	Number of chunks	Running time of the second model (sec)	Running time of the hypercube (sec)
Data 3	6	32.532	32.369
	12	17.861	16.800
	18	11.915	11.741
	24	7.954	7.247
Data 4	6	62.518	61.003
	12	31.206	31.060
	18	20.470	21.087
	24	14.722	13.854
Data 5	6	99.642	98.704
	12	49.350	48.208
	18	37.021	32.243
	24	26.077	24.011

7. CONCLUSION

The OpenMP parallel computing model analyzes the parallel performance of the given fractal models to figure out the parallel speedup and efficiency. When these different models used the MapReduce method to count the words, the results showed the following: i) the data results reveal that the first model performs marginally better than the second model, which is especially clear in the large data sets; ii) a well-known network model called the hypercube was used as a benchmark to assess how well the suggested models performed. The findings indicated a preference for the first network model offered, whereas the second network model proposed was quite similar to the first model regarding the percentage difference between them;

iii) more significant dynamic storage reduction, or better storage efficiency, was obtained using the storage technique in conjunction with the suggested fractal models' linked lists; and iv) the path-finding algorithm used in the suggested fractal models accomplished a great job of discovering all pathways and determining the shortest and longest routes. Finally, considering the performance we saw in our experimental configuration, it is possible to increase the performance with more powerful machines and a more significant number of cores. Although it has limits in the parallel OpenMP context, using GPUs is another potential option for scaling performance.




REFERENCES

- [1] S. H. H. Madni, M. S. A. Latiff, Y. Coulibaly, and S. M. Abdulhamid, "Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges and opportunities," *Journal of Network and Computer Applications*, vol. 68, pp. 173–200, Jun. 2016, doi: 10.1016/j.jnca.2016.04.016.
- [2] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, vol. 143, pp. 1–33, Oct. 2019, doi: 10.1016/j.jnca.2019.06.006.
- [3] S. Heidari, M. Alborzi, R. Radfar, M. A. Afsharkazemi, and A. R. Ghatari, "Big data clustering with varied density based on MapReduce," *Journal of Big Data*, vol. 6, no. 1, pp. 1–16, Dec. 2019, doi: 10.1186/s40537-019-0236-x.
- [4] A. Wakde, P. Shende, S. Waydande, S. Uttarwar, and G. Deshmukh, "Comparative analysis of Hadoop tools and Spark technology," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCCUBEA)*, Aug. 2018, pp. 1–4, doi: 10.1109/ICCCUBEA.2018.8697577.
- [5] A. Chuan, B. Chen, L. Liu, J. Dong, L. Hey, and X. Qiu, "Design and implementation of information dissemination simulation algorithm in large-scale complex network based on Spark," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, Jun. 2018, pp. 457–464, doi: 10.1109/DSC.2018.00074.
- [6] F. Jenhani, M. S. Gouider, and L. B. Said, "Streaming social media data analysis for events extraction and warehousing using hadoop and storm: Drug abuse case study," *Procedia Computer Science*, vol. 159, pp. 1459–1467, 2019, doi: 10.1016/j.procs.2019.09.316.
- [7] V. Jain, S. Chouhan, and K. K. Goyal, "Analyzing heuristic job scheduling algorithms by varying cloudlet load in a cloud infrastructure," in *2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART)*, Dec. 2021, pp. 531–535, doi: 10.1109/SMART52563.2021.9676267.
- [8] W. Kwedlo and P. J. Czochochanski, "A hybrid MPI/OpenMP parallelization of K-means algorithms accelerated using the triangle inequality," *IEEE Access*, vol. 7, pp. 42280–42297, 2019, doi: 10.1109/ACCESS.2019.2907885.
- [9] B. A. Mahafzah, M. Alshraideh, L. Tahat, and N. Almasri, "Topological properties assessment for hyper hexa-cell interconnection network," *International journal of computers*, vol. 13, pp. 115–121, 2019.
- [10] M. Abd-El-Barr and F. Gebali, "Reliability analysis and fault tolerance for hypercube multi-computer networks," *Information Sciences*, vol. 276, pp. 295–318, Aug. 2014, doi: 10.1016/j.ins.2013.10.031.
- [11] P. V. Reddy, S. Jena, and V. K. Prasa, "An efficient exchanged hyper cube for parallel and distributed network," *International Journal of Recent Technology and Engineering*, vol. 8, no. 2S10, pp. 821–829, Oct. 2019, doi: 10.35940/ijrte.B1150.0982S1019.
- [12] S. Ramirez-Gallego, A. Fernández, S. Garcia, M. Chen, and F. Herrera, "Big data: tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce," *Information Fusion*, vol. 42, pp. 51–61, Jul. 2018, doi: 10.1016/j.inffus.2017.10.001.
- [13] N. Maleki, A. M. Rahmani, and M. Conti, "MapReduce: an infrastructure review and research insights," *The Journal of Supercomputing*, vol. 75, no. 10, pp. 6934–7002, Oct. 2019, doi: 10.1007/s11227-019-02907-5.
- [14] T.-C. Huang, K. Chu, G. Huang, Y.-C. Shen, and C.-K. Shieh, "Distributed control framework for MapReduce cloud on cloud computing," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2018, pp. 1–4, doi: 10.1109/NOMS.2018.8406180.
- [15] J. Klinkenberg, P. Samfass, M. Bader, C. Terboven, and M. S. Müller, "Chameleon: reactive load balancing for hybrid MPI+OpenMP task-parallel applications," *Journal of Parallel and Distributed Computing*, vol. 138, pp. 55–64, Apr. 2020, doi: 10.1016/j.jpdc.2019.12.005.
- [16] A. Afzal, C. A. Saleel, K. Prashantha, S. Bhattacharyya, and M. Sadhikh, "Parallel finite volume method-based fluid flow computations using OpenMP and CUDA applying different schemes," *Journal of Thermal Analysis and Calorimetry*, vol. 145, no. 4, pp. 1891–1909, Aug. 2021, doi: 10.1007/s10973-021-10637-1.
- [17] X. Peng *et al.*, "Parallel computing of three-dimensional discontinuous deformation analysis based on OpenMP," *Computers and Geotechnics*, vol. 106, pp. 304–313, Feb. 2019, doi: 10.1016/j.compgeo.2018.11.016.
- [18] J. Zhao and M. Zhang, "Refactoring OpenMP code based on MapReduce model," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Dec. 2018, pp. 1040–1041, doi: 10.1109/BDCloud.2018.00153.
- [19] A. Afzal, Z. Ansari, and M. K. Ramis, "Parallelization of numerical conjugate heat transfer analysis in parallel plate channel using OpenMP," *Arabian Journal for Science and Engineering*, vol. 45, no. 11, pp. 8981–8997, Nov. 2020, doi: 10.1007/s13369-020-04640-1.
- [20] M. Aldinucci *et al.*, "Practical parallelization of scientific applications with OpenMP, OpenACC and MPI," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 13–29, Nov. 2021, doi: 10.1016/j.jpdc.2021.05.017.
- [21] P. Yu, X. Peng, G. Chen, L. Guo, and Y. Zhang, "OpenMP-based parallel two-dimensional discontinuous deformation analysis for large-scale simulation," *International Journal of Geomechanics*, vol. 20, no. 7, pp. 1–14, Jul. 2020, doi: 10.1061/(ASCE)GM.1943-5622.0001705.
- [22] X. Yang, W. Zhou, P. Zhao, and S. Yuan, "Confined electrons in effective plane fractals," *Physical Review B*, vol. 102, no. 24, pp. 1–10, Dec. 2020, doi: 10.1103/PhysRevB.102.245425.
- [23] K. Kavitha, "Design of a Sierpinski gasket fractal bowtie antenna for multiband applications," *International Journal of Applied Engineering Research*, vol. 13, no. 9, pp. 6865–6869, 2018.
- [24] F. Jahanmiri and D. C. Parker, "An overview of fractal geometry applied to urban planning," *Land*, vol. 11, no. 4, pp. 1–23, Mar. 2022, doi: 10.3390/land11040475.




- [25] M. Saltan, "Intrinsic metrics on Sierpinski-like triangles and their geometric properties," *Symmetry*, vol. 10, no. 6, pp. 1–12, Jun. 2018, doi: 10.3390/sym10060204.
- [26] S. Anarova, F. Nuraliev, and O. Narzullov, "Construction of the equation of fractals structure based on the rvachev r-functions theories," *Journal of Physics: Conference Series*, vol. 1260, no. 7, pp. 1–8, Aug. 2019, doi: 10.1088/1742-6596/1260/7/072001.
- [27] M. Yıldırım, F. Y. Okay, and S. Özdemir, "Big data analytics for default prediction using graph theory," *Expert Systems with Applications*, vol. 176, pp. 1–17, Aug. 2021, doi: 10.1016/j.eswa.2021.114840.
- [28] X. Mei, X. Cai, L. Yang, and N. Wang, "Graph transformer networks based text representation," *Neurocomputing*, vol. 463, pp. 91–100, Nov. 2021, doi: 10.1016/j.neucom.2021.08.032.
- [29] R. Yarinezhad and M. Sabaei, "An optimal cluster-based routing algorithm for lifetime maximization of internet of things," *Journal of Parallel and Distributed Computing*, vol. 156, pp. 7–24, Oct. 2021, doi: 10.1016/j.jpdc.2021.05.005.
- [30] J. Li, H. Lu, K. Xue, and S. Member, "Temporal netgrid model-based dynamic routing in large-scale small satellite networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 6009–6021, 2019, doi: 10.1109/TVT.2019.2910570.

BIOGRAPHIES OF AUTHORS






Muslim Mohsin Khudhair    is a staff member in the Department of Computer Information Systems, College of Computer Science and Information Technology, University of Basrah, Basrah, Iraq. Currently a Ph.D student in the Department of Computer Science, Faculty of Computer Science and Mathematics, at the University of Kufa. He received the B.Sc. and M.Sc. degrees in computer science from the College of Science, University of Basrah, Basrah, Iraq. His areas of interest include wireless sensor networks, artificial intelligence, image processing, and applied mathematics. He can be contacted at email: muslim.khudhair@uobasrah.edu.iq or mos1970@yahoo.com.



Furkan Rabee    is a staff member in the Department of Computer Science, Faculty of Computer Science and Mathematics, at the University of Kufa. He got B.Sc. and M.Sc. in Computer Engineering from AL-Nahrian University in 2000 and 2008. He obtained Ph.D in Computer Science and IT from the School of Computer Science and Engineering, UESTC, Chengdu, China 2015. The research interests include real-time scheduling algorithms, real-time locking protocols, operating systems, parallel processing, distributed system, computer network, iot, mobile computing, cloud computing, and smart cities. He can be contacted at email: furqan.rabee@uokufa.edu.iq.



Adil AL_Rammahi    he has Ph.D from 2005 in fractal geometry and was awarded the title of Professor of Mathematics in 2014. Included in the official business are the head of the Department of Mathematics and the Assistant Dean for Administrative and Scientific Affairs as well. He has several publications in Scopus journals and research contributions to conferences in Los Angeles, London, Paris and Geneva. He has books on programming and functional analysis. He can be contacted at email: adilm.hasan@uokufa.edu.iq.