A Parallel Genetic Algorithm for Maximum Flow Problem

Ola M. Surakhi
Computer Science Department
University of Jordan
Amman-Jordan

Mohammad Qatawneh
Computer Science Department
University of Jordan
Amman-Jordan

Hussein A. al Ofeishat
Computer Science Department
Al-Balqa applied university Jordan
Amman-Jordan

***Abstract*—The maximum flow problem is a type of network optimization problem in the flow graph theory. Many important applications used the maximum flow problem and thus it has been studied by many researchers using different methods. Ford Fulkerson algorithm is the most popular algorithm that used to solve the maximum flow problem, but its complexity is high. In this paper, a parallel Genetic algorithm is applied to find a maximum flow in a weighted directed graph, by finding the objective function value for each augmenting path from the source to the sink simultaneously in the parallel steps in every iteration. The algorithm is implemented using Message Passing Interface (MPI) library, and results are conducted from a real distributed system IMAN1 supercomputer and were compared with a sequential version of Genetic-Maxflow. The simulation results show this parallel algorithm speedup the running time by achieving up to 50% parallel efficiency.**

***Keywords*—Flow network; Ford Fulkerson algorithm; Genetic algorithm; Max Flow problem; MPI; multithread; supercomputer**

I. INTRODUCTION

A flow network is a directed graph where each edge has a capacity and receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge, and it must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, except when it is a source, which has more outgoing flow, or sink, which has more incoming flow [1]. The flow networks can represent many real-life situations like fluids in pipes for city water distribution, traffic in roads and more.

The maximum flow problem is one of the several well-known basic problems for combinatorial optimization in weighted directed graphs [2]. It involves finding a feasible flow from the source to the sink in a maximum flow network. The Ford-Fulkerson algorithm is the most widely used algorithm for solving maximum flow problem. The main idea of the algorithm is to find a path through the graph from the source (start node) to the sink (end node), in order to send a flow through this path without exceeding its capacity. Then we find another path, and so on. A path with available capacity is called an augmenting path [3], [4]. The time complexity of the Ford-Fulkerson algorithm is high. Therefore, a variety of researches have been applied to solve maximum flow problem using different methods and techniques [5].

In this paper, Genetic algorithm is applied in parallel to accelerate the process of finding the maximum flow problem and increasing the availability of high computer performance. Two conditions must be satisfied on the maximum flow problem: 1) The flow at each edge must not exceeds its capacity. 2) At each vertex, the incoming flow must be equal to the outgoing flow. The algorithm is implemented using MPI which is a standard library for message passing that can be used to develop portable parallel programs using C, C++ or FORTRAN [6], [7]. The evaluation is done in terms of the speed and parallel efficiency according to different network data size and different number of processors. The results were conducted using IMAN1 supercomputer which is Jordan's first and fastest supercomputer. It is available for the use of academia and industry in the region of Jordan. It provides multiple resources and clusters to run and test High Performance Computing (HPC) codes [7], [8].

The rest of this paper is organized as: Section 2 presents some related works to the maximum flow problem. Section 3 reviews the maximum flow problem. Section 4 introduces the sequential and parallel Genetic algorithm, and Section 5 presents the conclusion and future works.

II. RELATED WORKS

The maximum flow problem has been studied by many researchers because of its importance for many areas of applications, such as communication networks, Airline scheduling, computer sciences, electrical powers, tracks and more. Ford Fulkerson proposed the first pseudo code for solving maximum flow problem by finding the augmenting path [3], [4]. Other methods translate the maximum flow problem into maximal flow problem in layered network [9]. [10] introduced the push and re-label method which maintains a pre-flow and updates it through-push operations. The re-label operation perform the fine-grain updates of the vertex distances. Orlin [11] presents improved polynomial time algorithms for the max flow problem defined on a network with n nodes and m arcs, and shows how to solve the max flow problem in O(nm) time, improving upon the best previous algorithm due to [12] who solved the max flow problem in O(nm logm/(n log n) n) time. Genetic algorithm was also applied to solve max flow optimization problems. [2], each solution is represented by a flow matrix. The fitness function is defined to reflect two characteristics: balancing vertices and the saturation rate of the flow. Starting with a population of randomized solutions, better and better solutions are sought through the genetic algorithm. Optimal or near optimal solutions are determined with a reasonable number of

iterations compared to other previous GA applications. In [13], the CRO algorithm was implemented to solve the maximum flow problem. The proposed algorithm showed a better performance with a complexity of O(I E2), for I iterations and E edges.

III. Maximum Flow Problem

The flow network is a directed graph with two distinguished nodes; source and sink. Each edge between two nodes has a non-negative capacity and receives a flow where amount of flow on an edge cannot exceed its capacity as shown in Fig. 1.

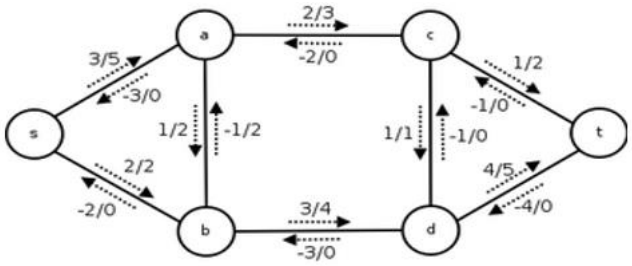

Fig. 1. An example of flow network [1].

For a directed graph G = (V, E), with source node S and a sink node T, and every edge e = (u,v) ∈ E has a non-negative, real-valued capacity c(u,v). The flow of the network is an integer valued function f that must satisfy following three properties for all nodes u and v:

1) Capacity constraints: f(u,v) ≤ c(u,v). The flow on each edge cannot exceed its capacity.

2) Skew symmetry: f(u,v) = −f(v,u). The flow from u to v must be the opposite of the net flow from v to u and f(u,u) =0.

3) *Flow conservation:* $\sum_{v \in V} f(s, v) = 0$,

4) the flow into a vertex must also flow out except for

5) the source, that "produces" flow, and the sink, which "consumes" flow.

The incoming flow to the node is equal to the outgoing flow from the node and thus the flow is conserved. Also, the total amount of flow going from source s equals total amount of flow into the sink t. the value of the flow is given by (1):

$$| f | = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t) \quad (1)$$

The maximum flow problem involves finding a flow from the source to the sink that is maximum to route as much flow as possible from s to t in the network.

IV. Sequential and Parallel Genetic Algorithm for Maximum Flow Problem

A. An overview of Genetic Algorithm

Genetic algorithm (GA) is a search based optimization algorithm inspired by the principle of genetics and natural selection. It begins with a population of possible solution to some problem which can be represented as a set of binary bit strings. Each individual in the population is assigned a fitness value based on its objective function value of the problem. The GA modified the population by applying the three main operations of it; reproduce, crossover and mutate to produce new children similar to natural genetic operators.

1) ***Reproduction*** *selects the best individual string from the population and discards the bad ones according to the fitness value. The best individuals are those having more chances to survive in the next generation.*

2) ***Crossover*** *includes two steps. First, select randomly two bit strings to be the parents of the new bit strings.*

Second, choose a place (crossover site) in the bit string and exchanges all characters of the parents after that point. The process tries to artificially mix the genetic of the parents and reproduce the mating process.

3) ***Mutation*** *changes the genes of the individual parents for the bits that didn't changed by the previous operations due to its absence from the generation, a 0 to 1 and vise versa.*

The genetic algorithm repeats these three operations until reaching the termination condition.

The pseudo code of GA is shown in Fig. 2.

B. Sequential GA for maximum flow problem

The GA has been applied to solve maximum flow optimization problems [2]. In [2], a flow matrix is used to represent each solution. The fitness function is defined to reflect two characteristics: 1) balancing vertices; and 2) the saturation rate of the flow.

```
(1) initialise population;
(2) evaluate population;
(3) while (!stopCondition) do
(4)     select the best-fit individuals for reproduction;
(5)     breed new individuals through crossover and mutation operations;
(6)     evaluate the individual fitness of new individuals;
(7)     replace least-fit population with new individuals;
```

Fig. 2. Generic pseudocode of a genetic algorithm [14].

Starting with a population of randomized solutions, the GA is applied for a reasonable number of iteration till reaching the optimal or near optimal solutions.

In this paper, a sequential implementation for GA is applied to find maximum flow problem with a different network size. The algorithm is implemented using Intel core I7-3632QM CPU2.20GHz, 8GB of RAM and windows 7 64 bits. The application programs were written in C language and executed on Net-Beans IDE 8.1. As mentioned before, the GA has three main operations; Reproduction (or Selection), Crossover and Mutation. The details of the implementation are discussed here.

For a graph, G with n vertices and m edges: G is represented by the flow capacity matrix, $C = [c_{ij}]$, i, j = 1, n.

Each solution is represented by a flow matrix $F = [f_{ij}]$, i, j = 1, n. The initial flow was generated randomly.

Selection step: There are different steps for selection. Through our approach, the probability to select some individual depends on its fitness value. We select half of all the individuals after calculating its fitness, then it will be ranked based on the fitness value, and from 0 to N/2 of the individuals will be selected.

Cross Over step: There are many ways to do crossover. Through our solution we make a cross over between selected population. We divide the population into two halves: F1 and F2. The crossover is done between the first half of F1 and the second half of F2 to produce S1, and crossover between first half of F2 with second half of F1 to produce S2, from this cross over new population was generated.

Mutation Step: A new population with full of individuals created after selection and crossover steps.

Some of them are directly copied, while others are produced by crossover. All the individuals should not be exactly the same. In order to ensure that a loop through all the alleles of all the individuals, and if that allele is selected for mutation, we can either change it by a small amount or replace it with a new value. The probability of mutation is usually between 1 and 2 tenths of a percent.

These steps will be repeated until reaching to maxflow value for selected generation. There fitness function used here is same as an objective function which is used to calculate Maxflow from source node to sink each iteration. These different steps will be repeated to select new population with new values for Maxflow from source to sink node.

The initialization step is important, through this step, different values must be defined and specified, like number of iteration, population size and mutation ratio. Number of iterations are important to achieve enhancement of solution at each iteration as GA is heuristic. Our experiment use different population size. The initial network size was 5000. The experiment was repeated by increasing the number of nodes, and stopped when it equals to 15,200 nodes as it consumes memory efficiency and space. The time needed to find max flow value is in seconds and shown in Table 1.

For the sequential implementation, the complexity depends on the population size and number of generations. And it can be defined as O(npg) where p is the population size and g is the number of generations.

C. Parallel GA for maximum flow problem

Finding the maximum flow value in a network graph can be done by running two main steps: 1) as long as there is a flow path from the source to the sink with a capacity c less than its flow value f, find this path; 2) change the flow accordingly. If no augmenting path exists, then we get the maximum flow. For a large network size with large number of nodes and arcs, dividing the graph into subgraph will enhance the running time needed to find the maximum flow value. In this case, the graph will be divided to a number of sub graph with a source and sink nodes for each one, every subgraph then, can be implemented in one processor to find its maximum flow value. The number of subgraphs will be equal to the number processors and the degree of concurrency will equal to the number of augmenting paths divided by number of processors as follows:

TABLE. I. TIME NEEDED FOR SEQUENTIAL GA TO FIND MAXFLOW VALUE WITH DIFFERENT NUMBER OF NODES

No. of nodes	Time/second
1000	0
2000	1
3000	1
4000	2
5000	3
6000	4
6300	4
7000	6
7700	7
8000	8
8300	9
8602	9
9000	10
10000	12
10400	14
11000	14
11400	15
11600	17
11800	21
12000	23
12200	29
12400	35
12800	39
13000	40
13400	50
14000	94
14200	147
14600	261
14800	343
15000	417
15200	480

Parallelism = total number of augmenting path/number of processors (2)

The implementation was done on Message Passing Interface (MPI) library, and results are conducted from a real distributed system IMAN1 supercomputer. The first implementation was done with one processor and then with two processors which reduced the time to half compared with the sequential time needed to solve maxflow problem as

shown in Fig. 3, then the number of processors were increased to 4, 8, 12, 16, 24 and 32, respectively. The initial network size 5000 and is increased repeatedly to reach 35,000 nodes. The implementation results are shown in Table 2.

The results show that using up to 4 processors in parallel can achieve a better result with a large network size, as the C language can measure the time with seconds only, we could not catch the enhancement in the running time when the number of nodes equals 5000 to 9000, the implementation gave an equal running time for 2 and 4 processors which could be less than the measured one if the estimated time was in millisecond. As the network size increased the running time reduced one or two seconds, a comparison between the running time for parallel maxflow-Genetic with 2 and 4 processors can be shown in Fig. 4.

TABLE. II. THE IMPLEMENTATION RESULTS FOR RUNNING MAXFLOW-GENETIC ON 1, 2, 4, 8, 12, 16, 24 AND 32

No of nodes	Time with 1-P	Time with 2-P	Time with 4-P	Time with 8-P	Time with 12-P	Time with 16-P	Time with 24-P	Time with 32-P
5,000	3	2	2	2	4	4	4	4
6,000	4	3	2	3	3	4	4	5
7,000	5	4	4	5	5	5	6	7
8,000	8	5	5	6	7	8	10	10
9,000	10	6	6	6	7	8	10	10
10,000	12	8	7	8	8	8	11	11
11,000	15	10	8	10	10	13	13	14
12,000	18	12	9	12	15	16	16	18
13,000	21	13	11	13	15	17	18	19
14,000	23	15	12	16	16	17	20	20
15,000	27	17	14	17	18	18	18	22
20,000	48	31	40	31	31	37	38	39
25,000	74	49	48	49	49	57	59	59
30,000	107	70	69	70	70	73	73	85
35,000	146	93	93	94	94	97	97	101

Using two-processors enhanced the efficiency of the system by reducing the running time to half as shown in Fig. 3.

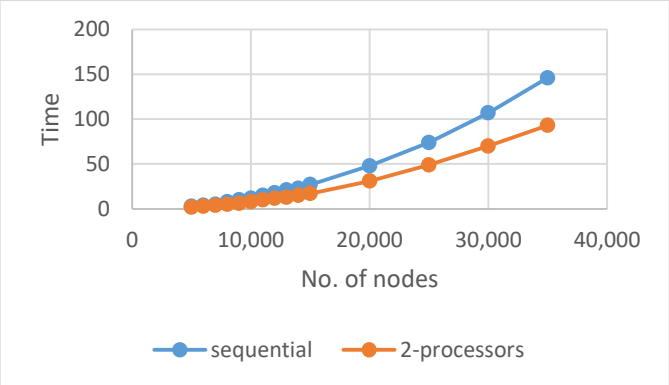

Fig. 3. Running time for parallel maxflow-Genetic with 1 and 2 processors.

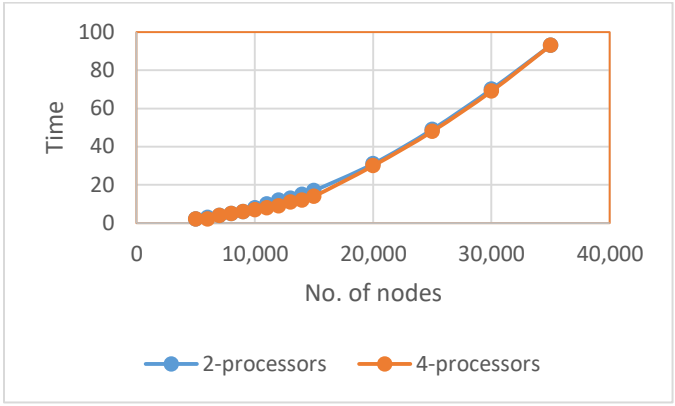

Fig. 4. Running time for parallel maxflow-Genetic with 2 and 4 processors.

Another important result could be noticed from Table 2. It shows that using more processors in parallel to solve maximum flow problem using GA could not give a better enhancement. It shows that using more processors in parallel to solve maximum flow problem using GA could not give a better enhancement. The speed up for this implementation is given in the following equation:

Speedup = sequential processing time/ parallel processing time (2)

Using (2) to find the speedup when using 2 and 4 processors give a result of 2. Fig. 5 shows the average speed up when using 1, 2, 4, 8, 12, 16, 24 and 32 processors.

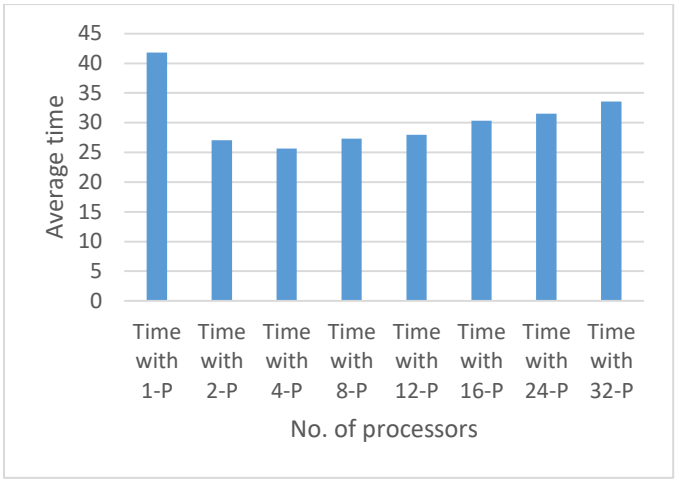

Fig. 5. Average speed up using 1, 2, 4, 8, 12, 16, 24 and 32 processors for parallel Maxflow-Genetic.

For 8, 12, 16, 24 and 32 processors, the running time increased by one second as the network size increased, which is close to the running time when using 4 processors. That's because of the communication between the processors to send and receive data. As the network size increase and the number of processors increase, the communication between the processors increased, which take more time than the time needed for execution. The running time for parallel maxflow-Genetic using 8, 12, 16, 24 and 32 processors are shown in Fig. 6.

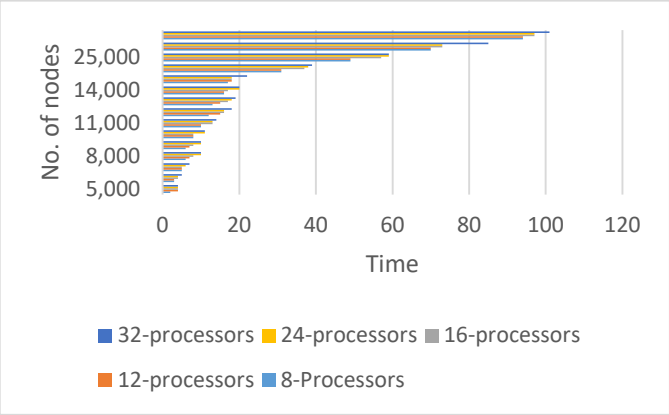

Fig. 6. Running time for parallel maxflow-Genetic with 8, 12, 16, 24 and 32 processors.

D. Parallel GA for maxflow problem in multi core processor

The parallel maxflow-Genetic has been applied on a multi core processors. That idea is similar to the parallel implementation on the distributed system, but in this case, the graph is divided over different number of threads, each of these threads work on separated core of CPU cores. Each subgraph has a set of augmenting paths, so each thread will calculate maximum flow value for its own nodes. The experiment was done with 2 threads, 4 and 6 threads respectively with a network size started initially with 5000 nodes and repeated 9 times till the number of nodes reached 12,000 with increasing by 1000 each time. The implementation was done using C programming language, on Intel Core I7-3632 QM CPU@2.20 GH with 8 GB internal memory.

The results show a better enhancement in the running time when compared with the time needed to find maxflow value with a sequential version of the maxflow-Genetic. The results are shown in Table 3 and Fig. 7.

TABLE. III. RUNNING TIME FOR MAXFLOW-GENETIC WITH 2, 4 AND 6 THREADS

No. of nodes	SEQ	2TH	4TH	6TH
5000	5	3	2	2
6000	6	3	3	3
7000	8	4	3	3
8000	9	5	4	4
9000	11	8	6	6
10000	13	9	7	7
11000	18	12	9	9
11500	23	16	16	12
12000	38	21	16	14

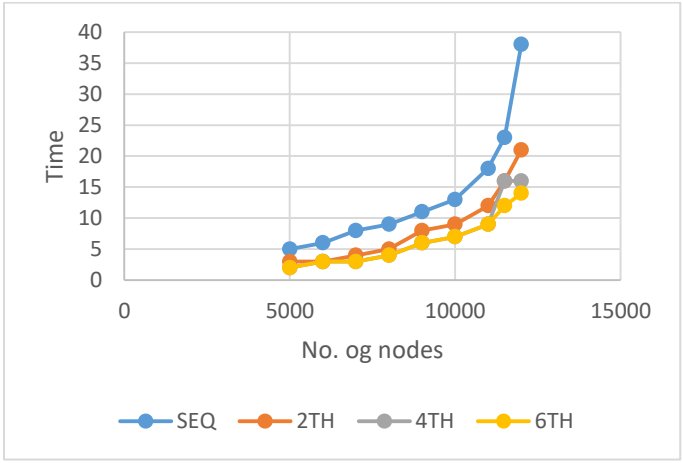

Fig. 7. Running time for maxflow-Genetic with 2, 4 and 6 threads.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, a parallel genetic algorithm has been implemented to solve maxflow problem. The implementation was done using open MPI library on IMAN1 supercomputer. The evaluation of the algorithm includes a different network size which starts from 5000 to 14,000 nodes. The results are compared with the sequential version of the algorithm and show a good enhancement in terms of the running time and system performance. Another implementation was done on a multi-core processor by dividing the graph into a set of subgraphs where each sub graph runs on its own thread. The results show a better enhancement in the running time when compared with the time needed to find maxflow value with a sequential version of the maxflow-Genetic.

As a future work, another heuristic, meta-heuristic or evolutionary algorithm could be used to find the maximum flow problem, like Chemical Reaction Optimization algorithm. The parallel implementation of the algorithm could be compared with the proposed one, and the results will be compared in terms of accuracy and performance.

REFERENCES

[1] Zhipeng Jiang, Xiaodong Hu, and Suixiang Gao, “A Parallel Ford-Fulkerson Algorithm For Maximum Flow Problem’.

[2] Munakata, T. and Hashier, D.J. “A genetic algorithm applied to the maximum flow problem”, Proc. 5thInt. Conf. Genetic Algorithms, 1993, pp. 488-493.

[3] Ford jr., L.R., Fulkerson, D.R., “Maximal flow through a network”. Can. J. Math. 8(3), 1956, pp. 399-404.

[4] L.R. Ford, Jr. and D.R. Fulkerson, Flows in Networks, Princeton, NJ: Princeton University Press, 1962.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, 3rd ed., The MIT Press, 2009.

[6] http://www.iman1.jo/iman1/, Accessed on (2014) July 10.

[7] M. Jeon and D. Kim, “Load-Balanced Parallel Merge Sort on Distributed Memory Parallel Computers”, Proceedings of the IEEE International Parallel and Distributed Processing Symposium, IPDPS.02, (2002).

[8] M. Saadeh, H. Saadeh, M. Qatawneh, "Performance Evaluation of Parallel Sorting Algorithms on IMAN1 Supercomputer" , International Journal of Advanced Science and Technology Vol.95 (2016), pp.57-72.

[9] Dinic, E.A., "Algorithm for solution of a problem of maximum flow in networks with power estimation". Sov. Math. Doklady. 11(8), 1970, pp. 1277-1280.

[10] R.K. Ahuja, T. L. Magnanti, and J. B. Orlin., Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.

[11] V. King, S. Rao, and R. Tarjan, "A faster deterministic maximum flow algorithm", In Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms, 1992, pp. 157–164.

[12] J.A. McHugh, Algorithmic Graph Theory, Englewood Cliffs, NJ: Prentice-Hall 1990, Chapter 6.

[13] R.Barham, A.Sharieh, A.Sliet. "Chemical Reaction Optimization for Max Flow Problem", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 7, No. 8, 2016.

[14] MahmoodA.Rashid,M.A.HakimNewton,Md.TamjidulHoque, AbdulSattar, "Mixing Energy Models in Genetic Algorithms for On-Lattice Protein Structure Prediction", Hindawi Publishing Corporation Bio Med Research International, Volume 2013,Article ID 924137,15 pages.