Genetic Algorithm for Data Exchange Optimization

Medhat H A Awadalla
Dept. of Electrical and Computer Engineering, SQU, Oman
Dept. of Communications and Computers, Helwan University, Egypt

***Abstract*—Dynamic architectures have emerged to be a promising implementation platform to provide flexibility, high performance, and low power consumption for computing devices. They can bring unique capabilities to computational tasks and offer the performance and energy efficiency of hardware with the flexibility of software. This paper proposes a genetic algorithm to develop an optimum configuration that optimizes the routing among its communicating processing nodes by minimizing the path length and maximizing possible parallel paths. In addition, this paper proposes forward, virtually inverse, and hybrid data exchange approaches to generate dynamic configurations that achieve data exchange optimization. Intensive experiments and qualitative comparisons have been conducted to show the effectiveness of the presented approaches. Results show significant performance improvement in terms of total execution time of up to 370%, 408%, 477%, and 550% when using configurations developed based on genetic algorithm, forward, virtually inverse, and hybrid data exchange techniques, respectively.**

Keywords*—Genetic algorithm; dynamic architectures; forward data exchange; virtually inverse data exchange; and hybrid data exchange method

I. Introduction

In recent years [2-3], the parallel architectures have obtained the popularity whether they are either fixed in their topology or more flexible in the way their architectures are constructed. These systems allow different amounts of resource sharing among its units depending on the way the units are interconnected. For one specific type of algorithms or problems, the static architectures can be designed to achieve a given requirements [4]. The arrangement of the units of the architecture reflects the algorithm/problem sort that the system tries to tackle. However, the dynamic architectures in the other side accommodate modular and adaptable components that can be controlled using automatic software to transfer the architecture from one state to another or from one configuration to another to fit different kinds of algorithms/problems, and this leads to improve the performance of the whole system. These dynamic architectures have links (paths) to be used to interconnect the architecture modules or resources and these links can be reconfigured under software control [5]. Changing the paths and the assignment of the modules and resources, the architecture can have different configurations/states. In each configuration/state, the modules and resources can be well selected to suit the specifications of the algorithm/application to get the maximum system performance [6].

The processors in the parallel architectures form a network and this network can be characterized by its topology or structure and it can be modelled as a graph. The nodes in this network (graph) represent the processors, the edges represent the links that are used to connect these nodes, and they are the means for data exchange among these nodes.

There are large number of parallel architectures that can be reconfigured and take different structures and topologies such as the architecture associated with multistage interconnection [7-8]. Even though the dynamic architectures provide flexibility to deal with different types of problems and contribute to the system performance improvement. However, any reconfiguration arrangement introduces two types of overhead, firstly, the reconfiguration hardware that needed to perform the reconfiguration process and not to do computation, control, or storage operations and secondly, the reconfiguration time that required to reconfigure the architecture from one configuration to another. It is so important in the early stages of designing dynamic systems to work out how to minimize these overheads.

In the literature, artificial intelligent techniques are widely used to solve problems that do not have definite conventional mathematical models for them. One approach of these artificial intelligent techniques is the Genetic Algorithm (GA). Many optimization problems have been solved using GA in different fields of Engineering and Science [9]. GA is a heuristic approach, which depends largely on random numbers to determine the approximate solution of an optimization problem. In the field of parallel and distributed systems, genetic algorithms have been used to address different algorithms and problems. Many authors proposed approaches to deal with the genetic operators [10-12]. Authors in [13] developed a genetic algorithm to reconfigure the topology and link capacities of an operational network in response to its operating conditions. The process of reconfiguration is very difficult if the addressed application/problem has many situations that can go through among them based on the different scenarios and situations. The authors in [14] proposed GA for autonomous architectural selection to find the best architectural configuration for the current situation. The assessment of their performance has been provided to illustrate that their approach efficiently found the best configuration [15]. The implementations of hardware genetic algorithms can also be observed in [16-17]. The authors presented their implementation of a genetic algorithm on FPGA that represented the population of chromosomes as a vector of probabilities. They tried to reduce the consumption of memory, power and space of the resources in hardware. In addition, the work in [18] proposed a high-speed GA implementation on FPGA, the authors claimed that their approach is the first implementation of GA on FPGA. They also claimed that their

developed system outperforms any existing or proposed solution related to their experiments.

In this paper, the capabilities of the genetic algorithm as an optimization technique have been utilized to find the optimum static structure to transfer data among processors connected together in a platform of multiprocessor. The structure should minimize the path length and maximize the possible parallel paths to ensure minimum time taken. In addition, the paper presents different approaches to generate dynamic configurations that enhance the system performance.

The rest of this paper is organized as follows. Section 2 presents the proposed genetic algorithm. Section 3 presents the generalized dynamic architecture. Section 4 shows the forward data exchange method. Section 5 presents the virtually inverse data exchange method. Section 6 presents the hybrid data exchange approach. Section 7 concludes the paper.

II. Proposed Genetic Algorithm (GA)

In this section, the main operators of the genetic algorithm have been described. Mainly, GA starts with initial population of chromosomes, which randomly generated and in some cases generated based on the output of another algorithm or an experiment. GA procedure is as follows.

1) The population should be initialized.

2) The population chromosomes should be assessed.

a) New chromosomes should be created using crossover and mutation operators.

b) Some of the existing population members (parents) should be deleted to give a room in the population for the new members (children/offspring).

c) The new members should be assessed and inserted into the population.

3) Step 2 should be repeated until termination condition is reached.

4) The achieved best chromosome is returned as the solution for the addressed problem.

A chromosome represents the solution of any problem tackled by Genetic Algorithm. The permutation of processor nodes P, $P \in V$ represents the chromosomes (V is the number of nodes) as shown in Table 1.

GA tournament selection operator is used in this paper to allocate the best trials to chromosomes according to the value of their fitness. Chromosomes are selected from the initial population to be parents for reproduction. In addition, elitism is used to keep the important information through the process of selection because they may not be selected through the GA operators, crossover and mutation and they get lost. A least the best two chromosomes are selected and placed into the mating pool, meanwhile are added in the next generation.

TABLE I. Processor Nodes Chromosome Representation

Gene	1	2	3	4	5	6	7	8
Chromosome 1	P0	P6	P2	P4	P3	P5	P1	P7

Tournament selection randomly picks a Tournament size (Ts) of chromosomes from the tournament that is a copy of the population (pop). The winner is best chromosome from (Ts) that has the best fitness (fit). This winner is then inserted into the mating pool (which is for example half of the tournament). The tournament competition is repeated until the mating pool for generating new offspring is filled. After that, crossover and mutation are performed. The developed tournament method is as shown in the following procedure.

Tournament Selection Method
tournamentselection (pop, fit, Ts);
BEGIN
1. Compute the size of the mating pool as size of population/2;
2. Compute the best two individuals from the population
3. Add them to the mating pool and the new population
4. for j←1 to Ts
5. DO compute random point as any point between 1 and population size
6. T[j] ← pop [point];
7. TF[j] ← fit [point];
8. END FOR
9. Compute the best one from T according to the fitness
10. Add it to the mating pool
11. Repeat steps 4 to 10 until mating pool is full
END

A. Crossover Operator

The crossover operator generates new chromosomes called children or offspring by combining two parent chromosomes. Based on the crossover probability pc, these chromosomes are exposed to single point crossover operator as shown in Fig. 1, otherwise, these chromosomes are not changed.

As shown in Fig. 1, after performing the crossover process, there are errors in representing the chromosomes where some processor nodes are presented twice in one chromosome. Chromosome 1 and chromosome 2 have duplicated the processor nodes P4 and P3 respectively. The problem is tackled using the following single-point crossover operator (Fig. 2). For any two randomly selected chromosomes c1 and c2, a cut point x is chosen randomly ($1 \leqslant x < V$). The first genes [1, x] of c1 and c2 are copied to the genes [1, x] of the new children ch1 and ch2 respectively. To fill the remaining genes [x + 1, V] of ch1 (ch2), chromosome c2 (c1) is scanned from the first to the last gene and each processor node that is not yet in ch1 (ch2) is added to the next empty position of ch1 (ch2) in the order that it is.

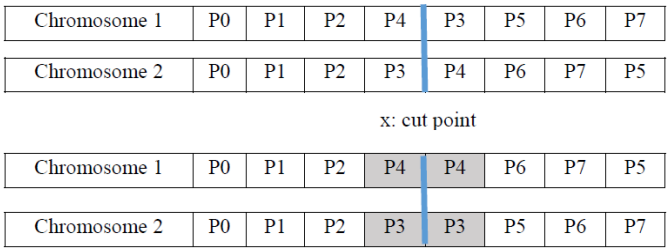

Fig. 1. Single-Point Crossover.

B. Mutation Operator

The probability of mutation operator (pm) is much less than that of the crossover operator. It is essentially for avoiding the convergence of a local solution. The mutation operator can be implemented through swapping randomly any two gens in a chromosome as shown in Fig. 3, where P5 and P6 are exchanged. During the simulated experiments, the population size was 100, the maximum number of generations was 500, the probability of the crossover was 0.7, and the probability of the mutation was 0.02. Table 2 presents some of the initial population chromosomes and the configurations based on the generated chromosomes are shown in Fig. 4, where the gens are arranged to represent the tree nodes of the top level first from the most left side to the right and then down towards the lower levels till reaching at the end to root node. For example, the chromosome C1 genes P7, P4, P6, and P1 are Level 3 nodes, P1 and P5 are Level 2 nodes, P3 represents Level 1 node, and P0 is the tree root.

In this paper, the Fitness Function is used to find the minimum total time required to exchange data among some processors as shown in Table 3.

$$FF = \min\left(\sum_{i=1}^{n} TE_i\right) \quad (1)$$

$$TE_i = ND_i * PL_i \quad (2)$$

Where, TE_i is the time needed for data exchange between two communicating processor nodes.

ND: the number of words to be transferred between the communicating processor nodes.

PL: the number of links between the communication processor nodes, assuming that the time required to transfer one word on one like is equal one unit of time, to make it simple assume it equal one.

n: the number of data requests to be transferred.

All configurations have been used to address the problem of data transfer between processors shown in Table 3 [19].

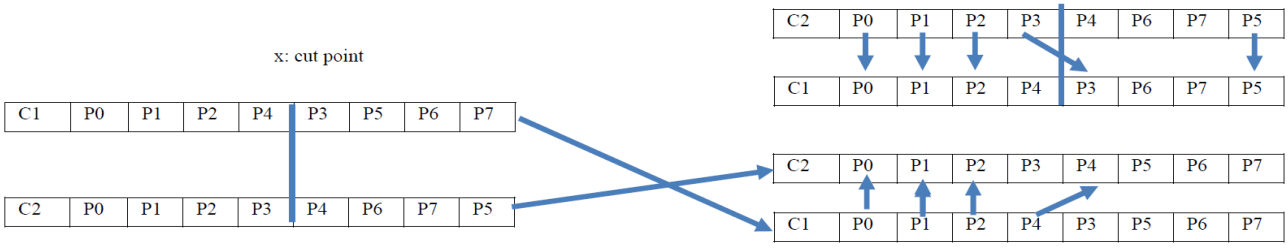

Fig. 2. The Developed Single-Point Crossover.

TABLE II. SOME OF INITIAL CHROMOSOMES

	Number of nodes							
Chromosomes	1	2	3	4	5	6	7	8
C1	P7	P4	P6	P2	P1	P5	P3	P0
C2	P4	P7	P6	P2	P1	P5	P3	P0
C3	P7	P4	P2	P6	P1	P5	P3	P0
C4	P7	P4	P6	P2	P1	P5	P0	P3
C5	P6	P2	P5	P7	P4	P1	P3	P0
C6	P6	P2	P5	P4	P7	P1	P3	P0
C7	P7	P4	P6	P2	P1	P3	P5	P0
C8	P7	P4	P6	P1	P2	P5	P3	P0
C9	P1	P7	P6	P4	P2	P5	P3	P0
C10	P2	P6	P5	P7	P4	P1	P3	P0
C11	P2	P6	P7	P5	P4	P1	P3	P0
C12	P2	P6	P5	P7	P1	P4	P3	P0
C13	P7	P4	P3	P2	P1	P5	P6	P0
C14	P1	P6	P5	P7	P2	P4	P3	P0
C15	P7	P4	P6	P0	P1	P5	P3	P2
C16	P7	P0	P6	P2	P1	P5	P3	P4
C17	P7	P6	P2	P4	P1	P5	P3	P0
C18	P6	P2	P7	P5	P4	P1	P3	P0
C19	P6	P2	P5	P4	P7	P1	P3	P0
C20	P1	P2	P3	P7	P4	P6	P0	P5

C1	P0	P1	P2	P4	P3	P6	P7	P5

Chromosome before mutation

C1	P0	P1	P2	P4	P3	P5	P7	P6

Chromosome after mutation

Fig. 3. Mutation Operator of Chromosome C1.

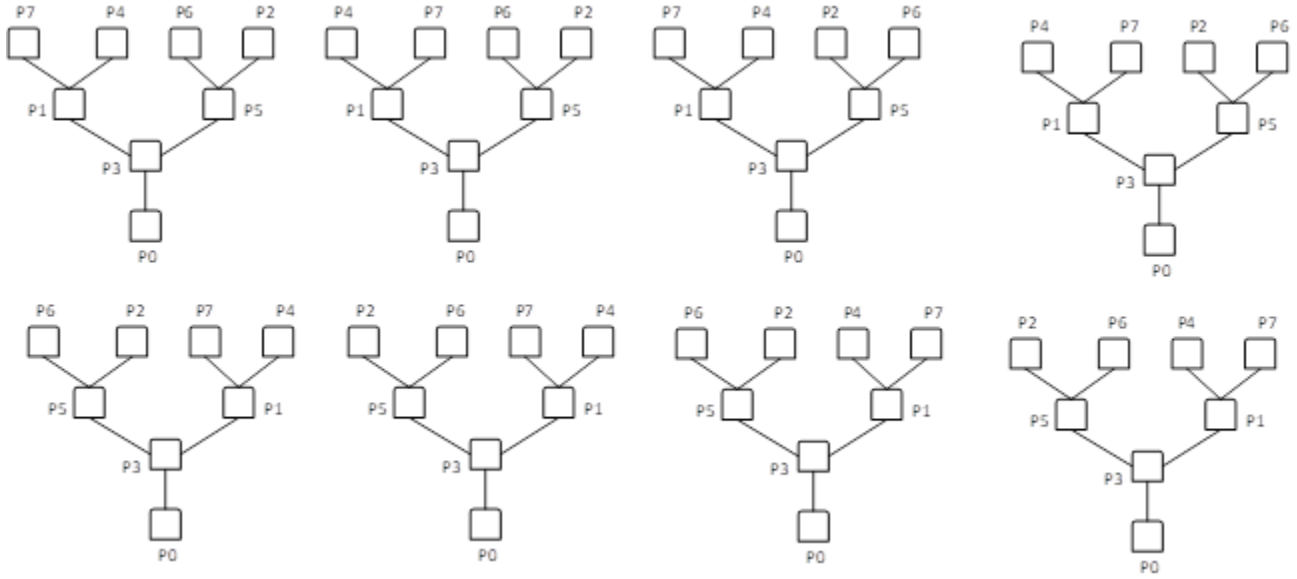

Fig. 4. Some of the Achieved Tree Configurations.

TABLE III. DATA EXCHANGE PROBLEM

Request No.	Source node	Destination node	Number of words to be transferred, ND	Length of minimal communication path in static binary tree, PL	Time of exchange in static tree T= ND*PL
1	3	2	50	3	150
2	7	1	100	4	400
3	6	4	20	1	20
4	5	2	100	1	100
5	4	0	50	1	50
6	6	2	30	2	60
7	3	5	100	4	400
8	1	3	100	4	400

After mating process as shown in Table 4 and other operators such as crossover and mutation are repeated until the end of all generations and reach to the stop criteria. In each time iteration, the fitness function is calculated. The fitness function for different configurations shown in Fig. 4 is illustrated in Table 5. One of the optimum static configurations achieved based on GA is depicted on Fig. 5, more than one configuration has the best fitness function value. Here, a brief explanation for determining the value of the fitness function is demonstrated. To conduct the data transfer between the processors shown in Table 3 on the tree configuration in Fig. 5, data exchange requests, 7-->1, 5 -- > 2, 3 -- > 5, and 1 -- >3 can be done in parallel. The total time to transfer all these data requests is equal the longest time of them. Since the requests are equal in number of data to be transferred and the number of links among them. Therefore, the total time is equal the time required to implement one request of them which is 100. The other requests will be transferred sequentially after the previous data exchanges such as 3 -- > 2 that takes 100 and then 4 -- > 0 that takes 150. The time needed for these data exchange is equal the longest of them, 150. The last two requests 6 -- > 2 and 6 -- > 4 are executed sequentially. The time required for them are 60 and 80, respectively. Then, the total time required to perform the whole job is equal 390, (100+150+60+80=390).

TABLE IV. THE DEVELOPED MATING POOL, 10 CHROMOSOMES

Chromosomes		Using elitism, the best two chromosomes are used in mating pool									
Mating	Population	1	2	3	4	5	6	7	8	fitness	
C1	**C1**	P7	P4	P6	P2	P1	P5	P3	P0	390	**best**
C2	**C2**	P4	P7	P6	P2	P1	P5	P3	P0	390	**best**
C3	**C3**	P7	P4	P2	P6	P1	P5	P3	P0	390	**best**
C4	**C4**	P7	P4	P6	P2	P1	P5	P0	P3	580	
C5	**C7**	P7	P4	P6	P2	P1	P3	P5	P0	580	
C6	**C17**	P7	P6	P2	P4	P1	P5	P3	P0	530	
C7	C13	P7	P4	P3	P2	P1	P5	P6	P0	610	
C8	C15	P7	P4	P6	P0	P1	P5	P3	P2	620	
C9	C16	P7	P0	P6	P2	P1	P5	P3	P4	390	best
C10	C9	P1	P7	P6	P4	P2	P5	P3	P0	640	

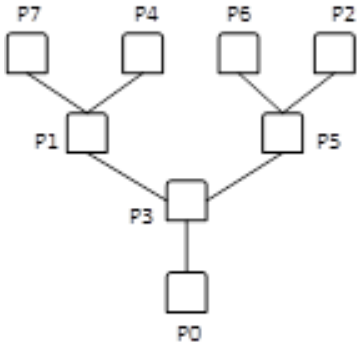

Fig. 5. GA based Optimum Static Configuration.

The time to perform the same problem by the best but not optimum tree constructed in [19] is also one of the configurations achieved by GA, and represented in Table 5 (C20) was 1430. Hence, there is a performance improvement of 370% using Genetic Algorithm. On the static tree shown in Fig. 6, the data exchange requests in Table 3 have been executed. This tree in Fig. 6 can be constructed in reality but it cannot be proved to be the optimum configuration because it is neither maximizing the concurrency nor minimizing the inter-processor communication. On this tree, out of eight data exchange requests, requests 1, 2, 3, 4, 6, and 8 are conducted sequential and just two requests 5 and 7 are accomplished in parallel. This is the reason behind the big amount of time required to conduct the total requests on this configuration. On the other side, GA has its limitations, the time for GA processes can be not omitted to find the optimum tree configuration even it is done offline.

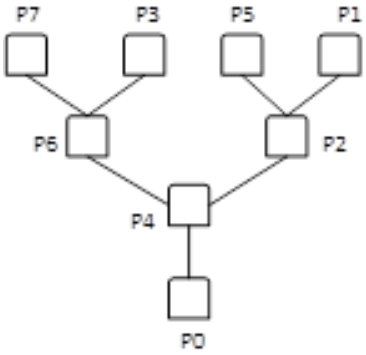

Fig. 6. Best Static Configuration in [20] and C20.

TABLE V. THE ACHIEVED FITNESS FUNCTION VALUES

Chromosomes	FF
C1	390
C2	390
C3	390
C4	580
C5	930
C6	1130
C7	580
C8	870
C9	640
C10	800
C11	700
C12	1160
C13	610
C14	1060
C15	620
C16	390
C17	530
C18	760
C19	1130
C20	1430

III. GENERALIZED DYNAMIC ARCHITECTURE

Even though the constructed static configuration is considered as the optimum structure because it gives the minimum time, the process of data exchange is accomplished in more than one phase sequentially as shown in Fig. 7. For large-scale problems, many data transfer will be executed sequentially and the number of links will be big which negatively affects the total required time for data exchange among the processors. The main aim of using dynamic configurations is to overcome the restrictions of the static structure. In this case, more than one configuration will be constructed and used to perform the data exchange to achieve a better performance compared with the performance of the optimum static configuration. The system develops the configuration that contains the longest communication node pairs are adjacent and conduct the data transfer among the source and destination nodes. After that, the system architecture will be reconfigured for the next longest communicating pairs to be adjacent to achieve the best performance; the developed algorithm is as follows:

Algorithm-1

1. Divide the requests into groups of equally number of data words need to be transferred.
2. Sort the groups in descending order according to the number of data.
3. For each group, if there is a destination node of any request is a source node for another request. Then, let this node is the intersection between these two requests.
4. Starting by top group, choose the configuration that maximizes the number of data exchange requests and minimizes the time needed for data exchange.
5. Assign all possible requests that can be executed concurrently.
6. Delete the assigned requests in step 5 from the data exchange table.
7. Repeat the previous steps until all requests will have been finished.

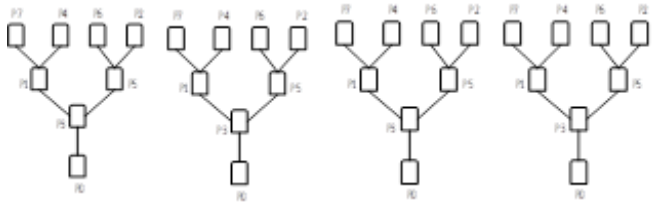

Fig. 7. The Implementation of Data Exchange on the Optimum Static Configuration.

To implement the developed algorithm, a dynamic architecture that can be reconfigurable is needed. Multistage interconnection networks addressed in [20-21] have been used due to their reconfiguration property. Such kind of an architecture can be reconfigured through software code. Multistage interconnection network has k stages and it can be used to connect n nodes ($n = m^k$), m and k are integers greater than one as shown in Fig. 8. If m is represented as $m = 2^{\alpha}$, so each node can be addressed by αk-bit binary number. Each stage element is controlled by a set of control lines and all switching elements in a certain stage receive the same control code and hence switch to the same state. If the inputs and the outputs of the switching elements are denoted by α-bit code as $i_{\alpha-1}\, i_{\alpha-2} \dots i_0$ ($j_{\alpha-1}\, j_{\alpha-2} \dots j_0$), each input node will be connected to the corresponding output node using the logic of Exclusive-OR between the input node and the control code as follows:

$$j_{\alpha-1}\, j_{\alpha-2} \dots j_0 = (i_{\alpha-1} \oplus c_{\alpha-1})\ (i_{\alpha-2} \oplus c_{\alpha-2}) \dots (i_0 \oplus c_0) \quad (3)$$

Where, $c_{\alpha-1}\, c_{\alpha-2} \dots c_0$ is α-bit control code that controls the state of each switching element. The switching states of the first S0 is different because there are m inputs and m2 outputs. Each input i is connected to output j as given by:

$$j = m * c + i \quad (4)$$

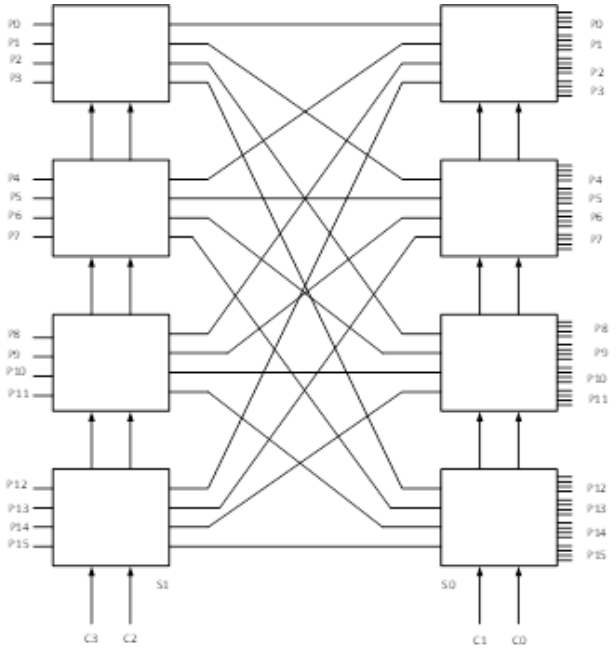

Fig. 8. The Reconfigurable Architecture with 16 Nodes.

Where, c is the decimal equivalent of α-bit binary control code, also there are m nodes are connected to the output of S0 and outputs of S0 are divided into m groups each m lines as a set for each node. The nodes to be connected to the inputs of S0 can be determined based on the following relation.

$$j_{\alpha-1}\, j_{\alpha-2} \dots j_0 = c_{\alpha-1}\, c_{\alpha-2} \dots c_0 \quad (5)$$

Referring to multistage interconnection network, a distinct tree structure with m^k nodes, is defined as a tree having k+1 levels (L_0, L_1, …, L_k) of nodes with root L_0 and leaf nodes at L_k. Each node at a level L_x (x=1, 2… k-1) is connected to m nodes at L_{k+1}. The root node is connected to m-1 nodes at L1 and has one connection with itself. The configuration control is responsible for establishing the configurations.

For each issued αk-bit control code, a distinct tree configuration is obtained. Each node $P(i)$ establishes a connection with a node $P(j)$ based on the following equation.

$$P(j) = CRS^{\alpha}(P(i)) \wedge B) \oplus \mathrm{C} \quad (6)$$

Where $CRS^{\alpha}(P(i))$ is α-bit the circular shift right of $P(i)$ and B is αk-bit number represented as 1α 1α … 0α.

Equation (6) can be rewritten to determine the required control code that if it is issued by configuration control, the tree that contains the adjacent $P(i)$ and $P(j)$ will be obtained as follows:

$$C = CRS^{\alpha}(P(i)) \wedge B) \oplus \mathrm{P}(j) \quad (7)$$

For instance, if m=2, k=3, and C=010, the processor nodes in the multistage interconnection network shown in Fig. 9 will form a tree as shown in Fig. 10. Fig. 11 shows the different configurations could be obtained from the dynamic architecture with different 3-bit control codes.

The next step is taking the next biggest data exchange, which is the data exchange between processor 5 and processor 2. In this case, the control code is 100 and the requests that can be executed concurrently are shown in Table 7 on the configuration achieved in Fig. 13.

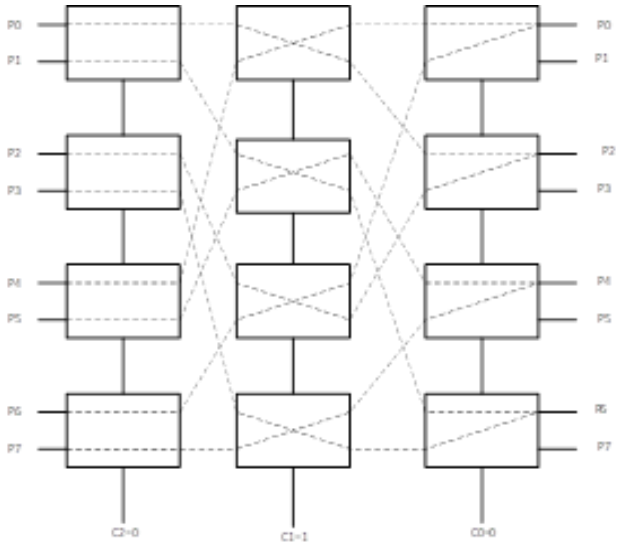

Fig. 9. The Reconfigurable Architecture with Eight Nodes.

TABLE VI. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
2	7 ==> 1	100
3	6 ==> 4	40
8	1 ==> 3	100
Total time		100

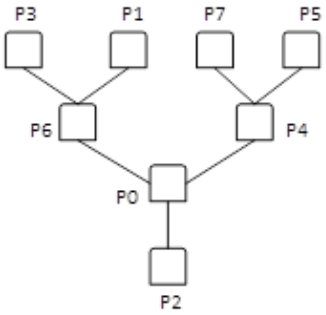

Fig. 10. The Formed Tree Topology m=2, k=3, c=010.

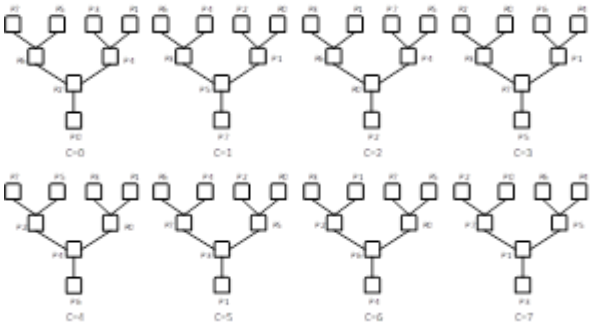

Fig. 11. Different Configurations with Different Control Codes.

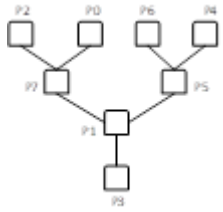

Fig. 12. The Formed Topology, C=111.

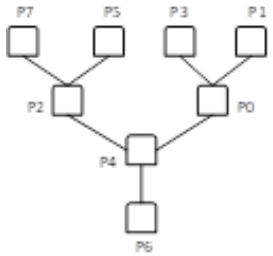

Fig. 13. The Formed Topology, C=100.

The next data exchange is between processor 3 and processor 5, for this case, the control code C is 001 is required. In this case, there is only one data exchange path as indicated in Table 8 and it will be accomplished through the configuration achieved in Fig. 14.

TABLE VII. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
4	5 ==> 2	100
5	4 ==> 0	50
6	6 ==> 2	100
Total time		100

TABLE VIII. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
7	3==> 5	100
Total time		100

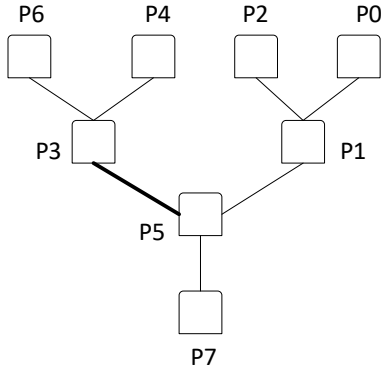

Fig. 14. The Formed Topology, C=001.

IV. PROPOSED FORWARD DATA EXCHANGE METHOD

Based on the developed algorithm-1 and using the dynamic configurations that can be achieved from the dynamic architecture through different software control codes, data exchange problem in Table 3 can be conducted. The heaviest communicating nodes should start communication first. If there are more than one pair, the lowest index will be considered in the data exchange problem in Table 3. The lowest index is defined as the top one of the communication path number that have the same number of words to be transferred in the table. For the problem under consideration, the first heaviest data exchange is between processor 7 and processor 1. Using equations 6 and 7, the required control code is determined as:

$$C = CRS^{1}(111) \wedge 110) \oplus 001 = 111 \quad (8)$$

When the configuration control unit issues this command, the tree structure in Fig. 12 will be formed. With the data exchange between processor 7 and processor 1, there are a possibility for some requests to be executed concurrently such as processor 6 and processor 4 as well as processor 1 and processor 3 and it is indicated in Table 6.

The last data exchange is between processor 3 and processor 2 and required control code C is 110 as shown in Table 9 and Fig. 15.

The total execution time is calculated from the above reconfiguration states as:

Total execution time = T (Ts1) + T (Ts2) + T (Ts3) + T (Ts4) =100+100+100+50=350

TABLE IX. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
1	3==> 2	50
Total time		50

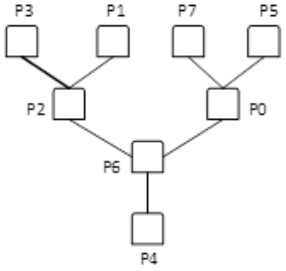

Fig. 15. The Formed Topology, C=110.

Of course, there is a time waste to generate new configurations, however for huge data to be transferred, processed, and using nowadays very fast computers, these factors can remarkably reduce that time. The improvement in the performance related to the developed genetic algorithm based static configuration is 112%. Comparing the achieved results with [20], the performance improvement is 408%.

V. PROPOSED VIRTUALLY INVERSE DATA EXCHANGE METHOD

It clear that, for any adjacent node pair in any given tree configuration, there is another tree that contains the same adjacent node pair but the difference is the instantaneous reverse direction of the data exchange. In the forward data exchange method, trees are built by considering that the data exchange between each pair of nodes takes place from a source node a destination one. In this method, trees are constructed by considering that the data exchange will be in the opposite direction, from a destination node to the source node.

Below is the description of reconfigurable structure of binary tree followed by assessment of the performance improvement that can be accomplished based on the inverse direction of data exchange. In this case, the reconfiguration equations take another form as:

$$P(i) = CRS^1(P(j)) \wedge B) \oplus \mathrm{C} \quad (9)$$

$$C = CRS^1(P(j)) \wedge B) \oplus \mathrm{P}(i) \quad (10)$$

Repeating of the above scenarios yields to the real execution from processor 7 to processor 1 has to be imagined as from processor 1 to processor 7. The deduced control code is given by:

$$C = CRS^1(001) \wedge 110) \oplus 111 = 011 \quad (11)$$

When the configuration control unit issues the control code, a tree that contains the processor node 7 and processor node 1 adjacent is constructed as shown in Fig. 16. All possible data exchange requests that can be performed concurrently with the data exchange between processor 7 and processor 1 are indicated in Table 10.

The next biggest data exchange is between processor node 5 and processor node 2, this is assumed to be from processor 2 to processor 5. In this case, the control code C is 101. Table 11 shows the possible requests that can be conducted on the achieved tree shown in Fig. 17 in parallel with the data exchange between processor 5 and processor 2.

The last heaviest data exchange is between processor 4 and processor 0. The C is 010. Table 12 shows the data exchange and all possibilities of data exchange that can be done in parallel with data exchange between processor 4 and processor 0 on the tree configuration in Fig. 18.

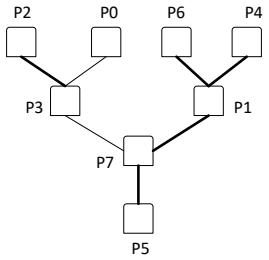

Fig. 16. The Formed Topology, C=011.

TABLE X. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
2	7 ==> 1	100
1	3 ==> 2	50
3	6 ==> 4	40
Total time		100

TABLE XI. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
4	5 ==> 2	100
7	3 ==> 5	100
8	1 ==> 3	100
Total time		100

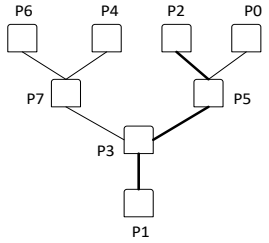

Fig. 17. The Formed Topology, C=101.

TABLE XII. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
5	4 ==> 0	50
6	6 ==> 2	60
Total time		60

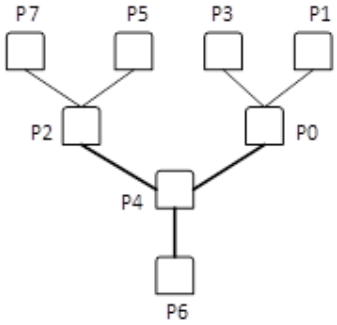

Fig. 18. The Formed Topology, C=010.

VI. Proposed Hybrid Method for Data Exchange

Although the virtual inverse method achieved an improvement in the system performance, in this new method, it is suggested to take the advantages of both forward and virtual inverse methods especially the cost of reconfiguration based on a few instructions to control the states of the switches. In order to find the sequence of the tree structures, which are used to conduct the data exchange, the following algorithm has been developed:

Algorithm-2

1. Search for the request (N_s, N_d) that have the maximum number of words to be transferred.
2. Find Ts1 and Ts2 based on the forward and virtual inverse methods respectively.
3. Assign all possible requests that can be executed concurrently.
4. Calculate the total number of data transferred and the time needed to transfer them.
5. Choose the configuration that maximizes the number of data exchange requests and minimizes the time needed for data exchange.
6. Delete the assigned requests in step 3 from the data exchange table.
7. Repeat the previous steps until all requests will have been finished.

Applying the developed algorithm in this method will develop the following configurations to conduct all data exchange requests. The first configuration is chosen based on the virtual inverse data exchange method. The completion of the data exchange requests is presented in Table 13 and Fig. 19.

The second configuration is chosen based on straight forward method. Where the control code is 100. Again, the completion of the data exchange requests is presented in Table 14 and Fig. 20.

The rest of data exchange requests have been executed on the configuration generated based on virtual inverse data exchange method. Where the control code is 101, the implementation is carried out on the configuration shown in in Fig. 21 and Table 15.

The total execution time to conduct all requests of data exchange equal 300. The improvement in the performance based on the hybrid method related to GA optimum static configuration is 130 %, and related to straightforward method is 117%, and the best static but not optimized structure is 477%. However, it cannot prove that it is better than virtual inverse method. In addition to the added overhead time of testing and comparing constructed configurations based on both methods time to choose the best of them. Using highly-speed processors, this method could manage to outperform the other methods especially for larger scale problems.

TABLE XIII. Possible Concurrently Data Change Requests

Satisfied requests	Data Path	Execution Time
2	7 ==> 1	100
1	3 ==> 2	50
3	6 ==> 4	40
Total time		100

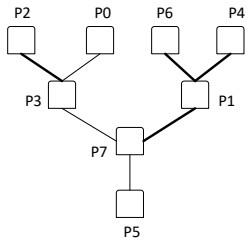

Fig. 19. The Formed Topology, C=111.

TABLE XIV. Possible Concurrently Data Change Requests

Satisfied requests	**Data Path**	**Execution Time**
4	5 ==> 2	100
5	4 ==> 0	50
6	6 ==> 2	60
Total time		100

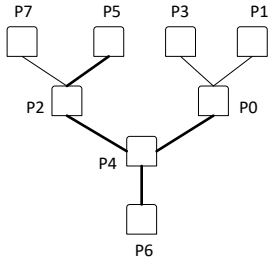

Fig. 20. The formed topology, c=100

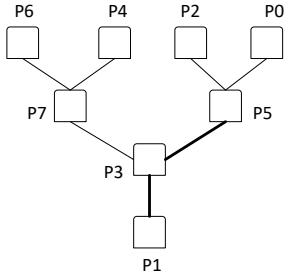

Fig. 21. The Formed Topology, c=101.

TABLE XV. POSSIBLE CONCURRENTLY DATA CHANGE REQUESTS

Satisfied requests	Data Path	Execution Time
7	3==> 5	100
8	1 ==> 3	100
Total time		100

VII. CONCLUSIONS

In this paper, genetic algorithm has been proposed to construct an optimum static configuration through which data exchange requests can be conducted. The achieved configuration was able to conduct many data exchanges in parallel and minimize the number of links between the communicated processors. However, due to the problem nature, not all of the requests can be accomplished in one shot. A sequence of dynamic configuration has been proposed to overcome the problem of static configuration. Forward, virtually inverse, and hybrid data exchange methods have been proposed to generate dynamic configurations that achieve data exchange optimization. The achieved results showed that there are performance improvements in terms of the total tasks' execution time of 370%, 408%, 477%, 550% using configurations developed based on genetic algorithm, forward, and virtually inverse, and hybrid data exchange techniques respectively.

REFERENCES

[1] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, K. Olukotun. "Plasticine: A Reconfigurable Architecture for Parallel Patterns". ISCA '17, June 24-28, 2017, Toronto, Canada, pp. 1-14.

[2] M. Lorenz, L.Mengibar, E. SanMillan, L. Entrena, " Low Power Data Processing system With Self reconfigurable". Journal of Systems Architecture, 2017, 53(9), pp. 568-576.

[3] P. Khera, A. Kumar, S. Singh, and S. Semwal. "Reconfigurable Architecture: An Approach to Design Low Power Digital Signal Processor". International Conference on Methods and Models in Science and Technology (ICM2ST-10).2010, pp. 433-437.

[4] R. Tessier, K. Pocek, and A. DeHon. "Reconfigurable Computing Architectures". Proceedings of the IEEE | Vol. 103, No. 3, 2015, pp. 332-354.

[5] M. Gao and C. Kozyrakis. "HRL: Efficient and flexible reconfigurable logic for near-data processing". International Symposium on High Performance Computer Architecture (HPCA), pp. 126–137.

[6] N. Instruments. "Understanding parallel hardware: Multiprocessors, hyper-threading, dual-core, multicore and FPGAs". URL: http://www.ni.com/tutorial/6097/en/.

[7] J. Cardoso, M. Hübner. "Reconfigurable Computing: From FPGAs to Hardware/Software Co-design". Springer 2011 edition, November 26, 2014.

[8] Y. Arzilawati, O. Mohamed; H. Zurina Lun, K. Yeah. "Number of sage implication towards multistage interconnection network reliability". Advanced Science Letters, V. 24, No. 2, February 2018, pp. 1259-1262.

[9] D. Kim and S. Park. "Dynamic Architectural Selection: A Genetic Algorithm Based Approach". International Symposium on Search Based Software Engineering, 2009, pp. 59-68.

[10] F. Mengxu, T. Bin, FPGA implementation of an adaptive genetic algorithm". Twelvish International Conference on Service Systems and Service Management (ICSSSM), 2015, pp. 1–5.

[11] H. Qu, K. Xing, T. Alexander. "An improved genetic algorithm with co-evolutionary strategy for global path planning of multiple mobile robots". Neurocomputing 120, 2013, 509–517.

[12] L. Guo, A. I. Funie, D. B. Thomas, H. Fu, W. Luk, Parallel genetic algorithms on multiple FPGAs, ACM SIGARCH Computer Architecture News 43, 2016, pp. 86–93.

[13] D. Montana, T. Hussain, T. Saxena. "Adaptive reconfiguration of data networks using genetic algorithms". Proceedings of the Genetic and Evolutionary Computation Conference, 2002, pp. 1141–1149, San Francisco, CA, USA.

[14] L. M. Ionescu, A. Mazare, A. I. Lita, G. Serban. "Fully integrated artificial intelligence solution for real time route tracking" 38th International Spring Seminar on Electronics Technology (ISSE), 2015, pp. 536–540.

[15] H. Qu, K. Xing, T. Alexander. "An improved genetic algorithm with co-evolutionary strategy for global path planning of multiple mobile robots". Neurocomputing 120, 2013, pp. 509–517.

[16] F. Mengxu, T. Bin, FPGA implementation of an adaptive genetic algorithm, in: 2015 12th International Conference on Service Systems and Service Management (ICSSSM), IEEE, 2015, pp. 1–5.

[17] H. Merabti, D. Massicotte. "Hardware implementation of a real-time genetic algorithm for adaptive filtering applications". 27th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), 2014, pp. 1–5.

[18] M. Vavouras, K. Papadimitriou, I. Papaefstathiou. "High-speed FPGA-based implementations of a genetic algorithm". SAMOS'09. IEEE International Symposium on Systems, Architectures, Modeling, and Simulation, 2009, pp. 9–16.

[19] G. Racherla, S. Radhakrishnan, L. Sumners DeBrunner. "Parameterization of efficient dynamic reconfigurable trees". Journal of Systems Architecture, 2000, 46(10), pp. 951-954.

[20] S. Rajkumar, N K Goya. "Review of multistage interconnection networks reliability and fault-tolerance". IETE Technical Review 2015.

[21] A. O. Balkan, G. Qu, U. Vishkin. "Mesh-of-trees and alternative interconnection networks for single-chip parallelism". IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2009.