Engineering Science and Technology, an International Journal 44 (2023) 101454

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Hamiltonian path, routing, broadcasting algorithms for connected square network graphs

Burhan Selçuk *, Ayşe Nur Altintaş Tankül

Department of Computer Engineering, Karabuk University, 78050 Karabuk, Turkey

ARTICLE INFO

Article history:
Received 9 December 2022
Revised 30 April 2023
Accepted 29 May 2023
Available online 1 July 2023

Keywords:
Meshes
Hamiltonian path
Routing
Broadcasting

ABSTRACT

Connected Square Network Graphs (*CSNG*) in the study of Selcuk (2022) and Selcuk and Tankul (2022) is reconsidered in this paper. Although (*CSNG*) is a 2-dimensional mesh structure, the most important feature of this graph is that it is a hypercube variant. For this reason, this study focuses on development algorithms that find solutions to various problems for (*CSNG*) with the help of hypercube. Firstly, an efficient algorithm that finds the Hamiltonian path is given. Further, two different algorithms that perform the mapping of labels in graph and the unicast routing are given. Furthermore, the parallel process for mapping and unicast routing is discussed. Finally, guidelines are given for broadcasting algorithms.

1. Introduction

Graphs appear in computer science both as a mathematical model and as a data type. Hypercube graphs and their variants are frequently used in computer architecture. Hayes et al. used hypercube architecture to implement supercomputers [1], Nieminen et al. described pseudocubes using hypercubes, while psedocubes have similar properties with hypercubes [2], Chae et al. studied on general cubic graphs [3], Harary et al. presented comprehensive analysis of hypercube graphs [4] and Mao and Nicol studied on graphs named k-ary n-cubes [5], proposed a technique to define and analyze a variation of this topology. Many researchers have studied hypercube and its variants and obtained their topological properties. For example Folded Hypercube is studied by El-Amawy and Latifi [6] and properties of hypercube is investigated, Cahng and Chen [7] proposed the incrementally extensible folded hypercube as a new class and explained its properties while giving a simple routing algorithm. The topological structure of Crossed Cubes, which is another hypercube variant, is studied by Dong et al. [8] and a 3D mesh family is embedded in it, Efe [9] gave an alternative way for parallel architectures using Crossed Cubes, and Lai et al. proposed a new dynamic programming algorithm to apply multidimensional torus in it [10]. Besides, there is also a lot of work on hierarchical networks. Abd-El-Barr and Al-Somani [11] reviewed and compared different type of hierarchical interconnection networks and their topological attributes, on the other hand Ghose and Desai [12] introduce the hierarchical cubic network (HCN), a novel interconnection system for massive distributed memory multiprocessors. Hierarchical extended Fibonacci cubes (HEFC) is a new interconnection network structure proposed by Karci [13] and its properties are explained. Karci also used Fibonacci series to suggest new graphs which are hierarchically definable and sub-graphs of Hierarchic Cubic graphs [14]. Karci and Selcuk introduced new hypercube variants. These; Fractal Cubic Network Graph (FCNG) [15] using the fractal structures and Connected Cubic Network Graph (CCNG) [16] using hypercube shown in Fig 1.2, Fig 1.3. and Fig 1.4., and explained their properties while giving algorithms for these graphs. Connected Square Network Graphs introduced by Selcuk can be obtained using two different definitions, and this variant of hypercube is a Hamiltonian graph [17]. A new Hypercube-like Graph is introduced by Selcuk and Tankul which is another type of hypercube and is labeled with using gray code and the connectivity indexes are calculated [18].

Motivated by [15,16], this paper proposes new algorithms for *CSNG*. Section 2 gives information about *CSNG*, divide and conquer method, dynamic programming, the travelling salesman problem, the routing problem and the broadcasting problem. Section 3 introduces a new Hamiltonian path algorithm employing dynamic programming techniques. Additionally, algorithms are provided for mapping and routing, along with a discussion on the parallel process for these algorithms in Section 4. Finally, broadcasting algorithms are given for *CSNG* with the help of broadcasting algorithms for hypercube.

* Corresponding author.
E-mail addresses: bselcuk@karabuk.edu.tr (B. Selçuk), aysenuraltintas@karabuk.edu.tr (A.N. Altintaş Tankül).

https://doi.org/10.1016/j.jestch.2023.101454
2215-0986/© 2023 Karabuk University. Publishing services by Elsevier B.V.

2. Preliminaries

For the rest of the study, let u and v be vertices in a graph G. G is considered to be *CSNG* with vertex set $V(G)$ and edge set $E(G), G = (V, E)$. "$||$" indicates the concatenation of two strings. The Hamming distance is $\sum_{i=0}^{n-1}(a_i \oplus b_i)$ since $\oplus$ is bit-wise XOR operation. $S(2)$ is denoted a square in $2D$ space and the $2D$ coordinate system is given in Fig. 1.

Definition 1. *CSNG*: Let $CSNG(0,0) = S(2)$. $CSNG(k,m)$ can be defined in two steps.

Case I. Construction in one direction

(i) Suppose $\sum_{i=0}^{m} 2^i$ squares with common two nodes (an edge) are connected along the y-axis. This graph will be called a $CSNG(0,m)$. For example, the mesh structures given in Fig. 2 (a) and Fig. 3 are $CSNG(0,1)$ and $CSNG(0,2)$, respectively.

(ii) Suppose $\sum_{i=0}^{k} 2^i$ squares with common two nodes (an edge) are connected along the x-axis. This graph will be called a $CSNG(k,0)$. For example, the mesh structure given in Fig. 2-(b) is $CSNG(1,0)$.

Case II. Construction in two directions

(i) Suppose $\sum_{i=0}^{m} 2^i$ $CSNG(k,0)$s with a common edge (right and left edges) are connected along the y-axis. This graph will be called a $CSNG(k,m)$.

(ii) Suppose $\sum_{i=0}^{k} 2^i$ $CSNG(0,m)$s with a common edge (lower and upper edges) are connected along the x-axis. This graph will be called a $CSNG(k,m)$. For example, the mesh structure given in Fig. 4 is $CSNG(1,2)$.

2.1. Divide and Conquer

There are different ways of designing algorithms. One approach is the divide-and-conquer algorithm, which, as the name suggests, divides the problem into sub-problems to make it easier to solve the original problem. It solves sub-problems that are like a smaller version of the original problem and combines the solutions of the sub-problems to form the solution of the original problem [19]. The divide and conquer paradigm increases program modularity, often leading to simple and efficient algorithms. It has therefore proven to be a powerful tool for sequential algorithm designers. The divide and conquer algorithms exhibit recursive structure, where the sub-problems are typically smaller in size compared to the original problem. Consequently, the original problem calls itself recursively to solve these sub-problems. At each recursive level, the divide and conquer method involves three steps.

- Divide the problem into similar sub-problems,
- Conquer these sub-problems with using recursive method to solve,

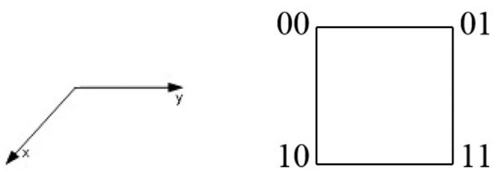

Fig. 1. a. $2D$ coordinate space, b. $S(2)$, respectively.

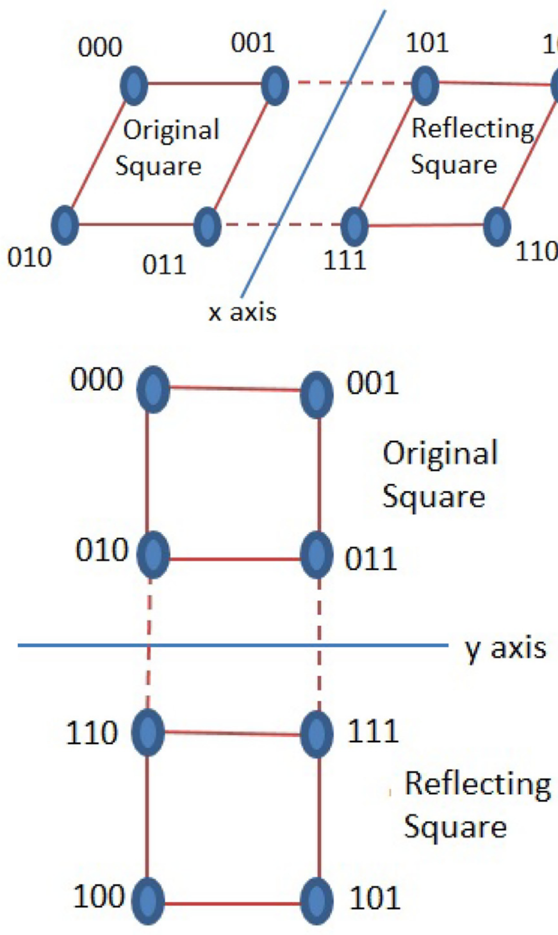

Fig. 2. a. $CCNG(0,1)$, b. $CCNG(1,0)$, respectively.

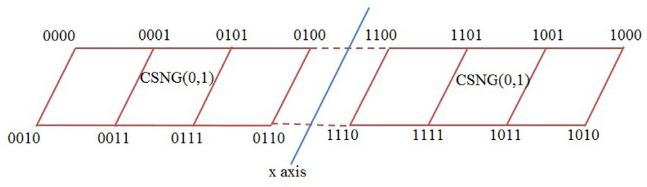

Fig. 3. $CCNG(0,2)$: Case 1-(i) of Def. 1.

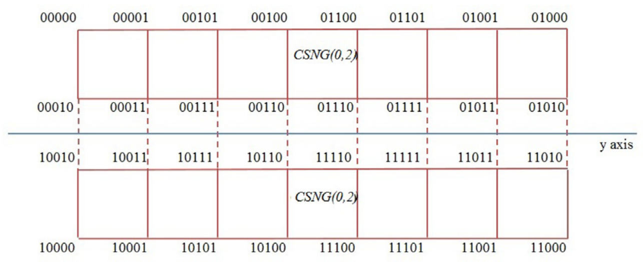

Fig. 4. $CCNG(1,2)$: Case 2-(ii) of Def. 1.

- Combine the solutions by solving the sub-problems into the solution that will be the original problem solution.

Parallel divide and conquer algorithm design plays an even more important role. Since the sub-problems created in the first step are usually independent, they can be solved in parallel. Usually the sub-problems are solved recursively, resulting in more sub-problems to be solved in parallel in the next dividing step. However, it should be noted that to obtain a highly parallel algorithm, the division and merge steps of divide and conquer must be parallelized. It is quite common in parallel algorithms to solve the original problem in parallel by dividing it into as many sub-problems as possible. For the Parallel Divide and Conquer example, consider the sequential Merge Sort algorithm. Merge sort takes n elements and returns them in order. This algorithm works by dividing an array of n elements into two parts, each dividing array recursively sorted, and combined the sorted sub-arrays.

Divide and conquer algorithm is implemented on various graph structures. On hypercube interconnection graph, a divide and conquer method to solve tridiagonal systems is implemented by [20,21] gives a generic divide and conquer implementation of the algorithms on hypercube like routing. On the other hand, in [22] divide and conquer mapping is proposed for efficient execution of parallel computers with hypercube structure. In [23], a binomial tree is embedded onto $3D$ mesh and torus interconnection network, and divide and conquer routing algorithm is proposed. A simple and optimal approach for mapping in $2D$ meshes is given in [24] and a partially adaptive and deterministic routing algorithm for $2D$ mesh networks on chip is given in [25]. Other mapping algorithm HyPar is proposed in [26] for hybrid GPU-CPU networks.

2.2. Dynamic Programming

Dynamic programming is generally applied to most of the optimization problems where a series of choices must be made to reach an optimal solution. Same sub-problems often arise while making choices. Dynamic programming is an algorithm design methodology much the same as divide and conquer, splitting problem into similar sub-problems and solving them recursively ([19]). Dynamic programming is used when the partitioned sub-problems of the original problem are repeated and not independent. Each sub-problem is solved just once by the algorithm and then the solution of the problem is stored in a table. By storing, it reduces the cost of code by eliminating the need to recalculate the same operations. The structure of dynamic programming is also recursive, a method calls itself for sub-problems of the main problem, and solves it if the sub-problem is encountered for the first time. Dynamic programming has four steps in development process [19];

- Characterization of the optimal solution structure,
- Iterative definition of the value for an optimal solution,
- Calculation of the value for an optimal solution from the bottom up,
- Building an optimal solution from the calculated information.

The first three steps are the basic steps of a dynamic programming. If the goal is solely to obtain the value of the optimal solution, it is sufficient to use only the first three steps and last step can be skipped.

A dynamic programming approach is used for layout optimization problem on interconnection networks [27], link scheduling [27] and routing [28] on mesh network, simulation of multi dimensional torus in crossed cube [10], optimal processor mapping for linear-complement communication [29], matrix chain product [30] and knapsack problem [31] on hypercube.

2.3. Travelling Salesman Problem (Hamiltonian path)

The Traveling Salesman Problem is one of the NP-hard optimization problem that means it has a large search space for the solution, so running time is not polynomial [32]. In the Traveling Salesman Problem (TSP), the goal is to find the shortest tour a seller takes starting from the city he is in and returning to the city where he started after stopping by each city only once. It is assumed that there is a road between any two cities and that the length of that road is known. It is easy to understand but difficult to solve. TSP, in the language of graph theory, is to find the shortest Hamiltonian cycle on a (plain) graph where cities are represented by the vertex and roads by the edges between the vertices.

The Hamiltonian path is named after 19th century mathematician William Hamiltonian which passes through each node on a graph only once. If a Hamiltonian path ends where it started, completing a full cycle (if it is possible to go from the last node on the path to the first node), these paths is called the Hamiltonian cycle. A graph containing a Hamiltonian cycle also contains a Hamiltonian path, but the reverse is not always true. The graphs that have a Hamiltonian cycle are called Hamiltonian graphs.

TSP is studied on interconnection topologies. Ant colony optimization algorithm applied on ring and hypercube in parallel for TSP [33], chemical reaction optimization [34] and grey wolf algorithm [35] is implemented for TSP over hypercube. [36] shows NISQ circuit compilation problem is same as TSP on torus network. TSP is also studied for mesh networks for channel allocation scheme for reducing interference [37].

2.4. Routing

Routing is the process of finding the shortest and potential paths between nodes in a network topology. There are two main types of routing problems, depending on their purpose and complexity [38]. The shortest path is the main focus of the path finding problems. Compared to other routing problems, these problems are relatively simple. Tour construction problems are the second group. It is more complex than the first category problems of building an entire tour within a given network. For instance, real distance and geographical information system (GIS) are used to solve real world problems represented as a network.

Path Finding Problems primarily encompass Shortest Path Problems (SPP), which involve determining the minimum total cost (time, distance, expense) between two nodes. Single Source Shortest Path Problem is for finding a path between source and destination vertices. There are various algorithms for these problems. All Pair Shortest Path Problem to find paths between all vertices pairs in the network. Tour construction problems can be divided into three classes: Travel Salesman Problem (TSP) is the first one and the most well known, Vehicle Routing Problem (VRP) is the generalized version of TSP with many salesmans, and lastly Bus Routing Problem (BRP) is another version of TSP with route balancing, may have time window constraints and busses can have same or different capacities.

SPP is implemented on many different types of interconnection networks. In [39] shortest path routing is applied for all to all communication on hypercube. In [40] average shortest distance is found with new model for irregular ring and hub network. One to one shortest path problem is studied for on-chip board large scale mesh network in [41], while one to all shortest path problem is studied for torus network [42].

2.5. Broadcasting

In computer networks, casting refers to sending data over the network used for communication between hosts. Types of casting used in networking are as follows [43]:

- Unicast transmission is one-to-one transmission used to send data from one sender to one receiver.

- Broadcast transmission is one-to-all transmission that sends data from one or more hosts to all hosts in the network.
- Multicast transmission is one-to-many transmission where there is a single source that sends data to the specific group in the network.

The broadcasting process occurs with sequential data transmissions between pairs of nodes. Broadcast communication is often required in image processing, parallel implementation of algorithms or scientific computations, spread large data arrays among system nodes and to perform various data processing activities. There are two way for broadcasting the message over the network [44]. First one is one-to-all broadcasting, transmission of the identical message from originating source to all nodes over the network. The m-size data that needs to be published is initially only available in the source node. At the end of the procedure there are n (number of nodes in the network) copies of the initial data, one at each node. All-to-all is the other type of broadcasting also called total exchange or gossip. In this broadcasting, each node sends a message to every other node on the network. All nodes start a broadcast simultaneously in an all-to-all broadcast, which is a generalization of the one-to-all broadcast. A node sends the same message to all nodes, however different nodes may broadcast different messages. Broadcasting algorithms are existed for sparse networks like torus and grid networks [45], mesh networks [46], star networks [47], and variant of hypercube networks [15,47–49]. The most known and used broadcast algorithms are Recursive Doubling, Network Partitioning and Extending Dominating Node algorithms [50].

3. Hamiltonian Path Algorithm

This section introduces a new Hamiltonian path algorithm for *CSNG*. The dynamic programming method is used in recursive structures like the divide and conquer method. Since it calculates the sub-problems once and stores this value in the table, it can offer better performance than the divide and conquer method. Thus, the dynamic programming method will be used in the following Algorithm 1. One bit change between labels of neighboring nodes is provided with the help of gray code in this algorithm.

Example 3.1. Firstly, it is assumed to be a Hamiltonian path for a square in 2D space;

$00 \to 01 \to 11 \to 10.$

Assume that initial node is 01001 on $CSNG(3,0)$ (or $CSNG(2,1), CSNG(1,2)$, $CSNG(0,3)$). In fact, 010 for the outer recursive structure and 01 for the inner recursive structure are chosen as the initial node. To obtain the Hamiltonian path for the inner recursive structure, the Hamiltonian path considered above must be shifted so that the starting node is 01 (See step 1 in Algorithm 1). A Hamiltonian path for the square will be named $H(0)$ and $I(0)$ (reverse sorted of $H(0)$) are obtained as follows:

$H(0) = 01 \to 11 \to 10 \to 00, I(0) = 00 \to 10 \to 11 \to 01.$

For the outer recursive structure (See step 2 in Algorithm 1), the last node is 0 on 010, and then the result of the first iteration;

$$\begin{aligned} H(1) &= 0||H(0) \to 1||I(0) \\ &= 001 \to 011 \to 010 \to 000 \to 100 \to 110 \to 111 \to 101, \end{aligned}$$

and

$$\begin{aligned} I(1) &= 1||H(0) \to 0||I(0) \\ &= 101 \to 111 \to 110 \to 100 \to 000 \to 010 \to 011 \to 001. \end{aligned}$$

The middle node is 1 on 010, and similarly the 2nd iteration can be obtained as follows.

$$\begin{aligned} &H(2) = \\ &=1001 \to 1011 \to 1010 \to 1000 \to 1100 \to 1110 \\ &\to 1111 \to 1101 \to 0101 \to 0111 \to 0110 \to 0100 \\ &\to 0000 \to 0010 \to 0011 \to 0001, \end{aligned}$$

and

$$\begin{aligned} I(2) = \;& 0||H(1) \to 1||I(1) \\ = \;& 0001 \to 0011 \to 0010 \to 0000 \to \\ & 0100 \to 0110 \to 0111 \to 0101 \to 1101 \\ & \to 1111 \to 1110 \to 1100 \to \\ & 1000 \to 1010 \to 1011 \to 1001. \end{aligned}$$

Start from node 0 on 010, and then the result of the last iteration;

$$\begin{aligned} H(3) =& 0||H(2) \to 1||I(2) = 01001 \to 01011 \to 01010 \to 01000 \to 01100 \to 01110 \\ & \to 01111 \to 01101 \to 00101 \to 00111 \to 00110 \to 00100 \to \\ & 00000 \to 00010 \to 00011 \to 00001 \to 10001 \to 10011 \to 10010 \\ & \to 10000 \to 10100 \to 10110 \to 10111 \to 10101 \to 11101 \to 11111 \\ & \to 11110 \to 11100 \to 11000 \to 11010 \to 11011 \to 11001. \end{aligned}$$

Remark 3.1. The Algorithm 1 finds Hamiltonian path for $CSNG(k,m)$ using dynamic programming. If this algorithm were designed with divide and conquer approach, two nested recursive structures would have to be encountered. By using dynamic programming instead of divide and conquer method, Hamilton path can be found with a simpler strategy. Step 1 does not include loop, recursion or other functions, only includes constant number of operations so its time complexity is $\mathcal{O}(1)$. For step 2, because it includes loop and the loop variable j starts from 1 and increments by one at each iteration until $k+m$ and includes "||" operations. This operation essentially contains a hidden for loop. For each j, the sum of the sizes of the arrays to be used in the "||" operation will give the complexity of the Algorithm 1. For $j=1$, size of array: 2^2, for $j=2$, size of array: 2^3,...,for $j=k+m$, size of array: 2^{k+m+1}. Total running numbers; $2^2+2^3+\ldots+2^{k+m+1} = 2^{k+m+2}-4$ using geometric series expansion. Thus, the complexity of the Algorithm1 is $\mathcal{O}(2^{k+m+2})$.

$CSNG(k,m)$ has 2^{k+m+2} nodes. $CSNG(0,0)$ is equivalent to the square and has 2^2 nodes. Although $C(2,0), C(0,2)$ and $C(1,1)$ have the same number of nodes, they are not isomorphic graphs. However, a Hamiltonian path for these graphs are not isomorphic due to the construction of the $CSNG(k,m)$ graph can be obtained in a similar way using the Algorithm1.

In $S(2)$, there are 8 Hamilton paths according to the starting address and node order, and there are 2 different Hamilton paths according to node order. These paths are specified as follows;

$00 \to 01 \to 11 \to 10$
$00 \to 10 \to 11 \to 01$
$01 \to 11 \to 10 \to 00$
$01 \to 00 \to 10 \to 11$
$11 \to 10 \to 00 \to 01$
$11 \to 01 \to 00 \to 10$
$10 \to 00 \to 01 \to 11$
$10 \to 11 \to 01 \to 00$

A general formula for these paths is given below using step 1 in Algorithm 1;

$$\begin{aligned}\text{Case } 1(H(0)):&\quad N\to N\oplus 2^0\to N\oplus 2^0\oplus 2^1\to N\oplus 2^1\\ :&\quad N\oplus 00\to N\oplus 01\to N\oplus 11\to N\oplus 10\\ \text{Case } 2(H(0)):&\quad N\to N\oplus 2^1\to N\oplus 2^1\oplus 2^0\to N\oplus 2^0\\ :&\quad N\oplus 00\to N\oplus 10\to N\oplus 11\to N\oplus 01\end{aligned}$$

where N(initialnode) : 00, 01, 11, 10. In order to derive new graphs by analogy with the hypercube strategy, a reverse order of the found paths is also needed. For this, the reverse paths are indicated as follows in step 1 in Algorithm 1;

$$\begin{aligned}\text{Case 1 } (I(0)):&\quad N\oplus 2^1\to N\oplus 2^1\oplus 2^0\to N\oplus 2^0\to N\\ :&\quad N\oplus 10\to N\oplus 11\to N\oplus 01\to N\oplus 00\\ \text{Case 2 } (I(0)):&\quad N\oplus 2^0\to N\oplus 2^0\oplus 2^1\to N\oplus 2^1\to N\\ :&\quad N\oplus 01\to N\oplus 11\to N\oplus 10\to N\oplus 00\end{aligned}$$

In step 2 of the Algorithm 1, the path obtained in the previous subproblem and the inverse of this path are combined using Definition 1.

CSNG(k,m) network structure is similar to 2D mesh network structure. In the literature, algorithms for Hamiltonian path in 2D mesh networks $(M(p,r))$ [51–53] with the size of pr have generally the time complexity $\mathcal{O}(pr)$. Here, $p=2^{k+1}$ and $r=2^{m+1}$. Thus, it is obtained that the proposed algorithm in *CSNG*(k,m) and the Hamiltonian algorithms for $M(p,r)$ have the same complexity.

Algorithm 1: This algorithm calculates an alternative Hamiltonian path with *CSNG*(k,m) using dynamic programming process.

```
Data: $k, m, N$: initial node and $case$: 1 or 2
Result: $H$: Hamiltonian path of $CSNG(k,m)$
1  begin
2      // Step 1: Alternative Hamiltonian path for square
3      if $k+m == 0$ then
4          if $case == 1$ then
5              $H(0) = N\oplus 00 \to N\oplus 01 \to N\oplus 11 \to N\oplus 10$
6              $I(0) = N\oplus 10 \to N\oplus 11 \to N\oplus 01 \to N\oplus 00$
7          if $case == 2$ then
8              $H(0) = N\oplus 00 \to N\oplus 10 \to N\oplus 11 \to N\oplus 01$
9              $I(0) = N\oplus 01 \to N\oplus 11 \to N\oplus 10 \to N\oplus 00$
10     // Step 2: A Hamiltonian path for $CSNG(k,m)$
11     for $j = 1$ to $k+m$ do
12         if $N(j+2) == 0$ then
13             $H(j) = 0||H(j-1) \to 1||I(j-1)$
               $I(j) = 1||H(j-1) \to 0||I(j-1)$
14         if $N(j+2) == 1$ then
15             $H(j) = 1||H(j-1) \to 0||I(j-1)$
               $I(j) = 0||H(j-1) \to 1||I(j-1)$
16     return $H$
```

4. Communication Algorithms

This section, introduces new unicast routing algorithms for *CSNG* and provides a recursive algorithm for mapping node labels of *CSNG*, as well as an iterative algorithm for finding path.

4.1. Unicast Routing Algorithms

The nodes on the networks decide which will receive the message sent by various methods while communicating with other nodes. Some messages are directly sent to a node, some are intended to reach a specific set of nodes, and some reach all nodes in the same network.

If the message sent to the network targets directly a single node which address is known, the message is sent using an algorithm designed as unicast. Thus, the network elements used in between perform the task of transmitting the message only to the specified node. Broadcast algorithms deliver the message to all nodes while multicast algorithms are used for communication within the group.

Unicast routing algorithm for *CSNG*(k,m) has a different strategy from e-cube routing algorithm for the hypercube. The classical method determines firstly how many bits are changed from source (S) to destination (D). In this study, a different strategy consisting of the following steps will be followed;

- *Step 1.* The labels of all the squares that construct *CSNG*(k,m) need to be calculated. The new labeling template for this graph is obtained by transforming Fig. 5 into Fig. 6 using Algorithm 2 (mapping algorithm),
- *Step 2.* The unicast routing path between source S and destination D nodes is obtained by traversing first rows (columns) and then columns (rows) from the corresponding squares with the help of the new labels of these nodes using Algorithm 3 (path-finding algorithm).

Example 4.1. Finding the map for the graph *CSNG*$(2,2)$ is a recursive process. There are 4 different base cases for Algorithm 2 according to the values of i and j. These cases are $(0,0),(1,0),(0,1)$ and $(1,1)$ for values of (i,j) that determines base case return map M and takes $[0001;1011],[0100;1110],[1011;0001],[1110;0100]$ values respectively for *CSNG*(k,m) graphs, so for the graph *CSNG*$(2,2)$ when k and m values are equal to 2. i is used for calculating the mirror inverse of the graph with respect to the y-axis, while j is used for calculating the mirror inverse of the graph with respect to the x-axis. Then, each time the function is called recursively, the size of the map is doubled. First, the map size is increased horizontally according to m values by merging *CSNG*$(0,m-1)$ and reversed order of the elements in each row of *CSNG*$(0,m-1)$ after adding 0 and 1 respectively at the beginning of each label. Then, the map size is increased vertically according to k values by merging *CSNG*$(k-1,m)$ and reversed order of the elements in each column of *CSNG*$(k-1,m)$ after adding 0 and 1 respectively at the beginning of each label. At the first call i and j must be 0. Step by step, the result of *Map*$(2,2,0,0)$:

M=[00 01;10 11]→

M=[000 001 101 100;010 011 111 110]→

M=[0000 0001 0101 0100 1100 1101 1001 1000; 0010 0011 0111 0110 1110 1111 1011 1010]→

M=[00000 00001 00101 00100 01100 01101 01001 01000; 00010 00011 00111 00110 01110 01111 01011 01010; 10010 10011 10111 10110 11110 11111 11011 11010; 10000 10001 10101 10100 11100 11101 11001 11000]→

M=[000000 000001 000101 000100 001100 001101 001001 001000; 000010 000011 000111 000110 001110 001111 001011 001010; 010010 010011 010111 010110 011110 011111 011011 011010; 010000 010001 010101 010100 011100 011101 011001 011000; 100000 100001 100101 100100 101100 101101 101001

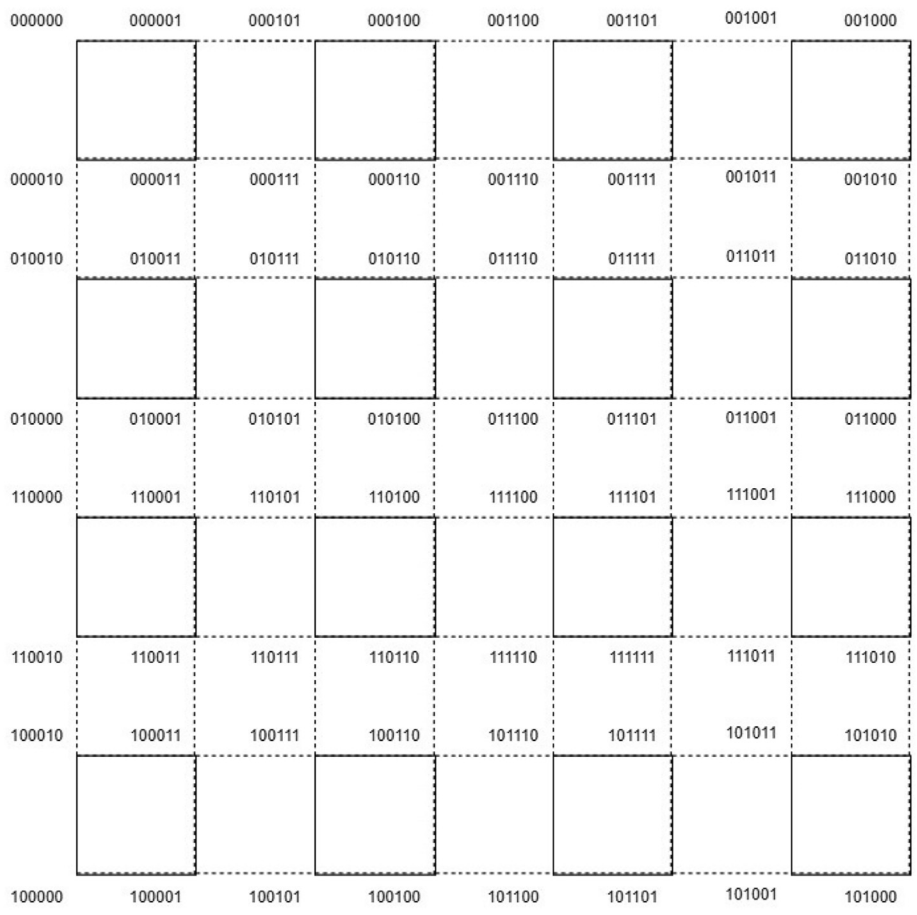

Fig. 5. Labelling of $CSNG(2,2)$ in [17].

101000; 100010 100011 100111 100110 101110 101111 101011 101010; 110010 110011 110111 110110 111110 111111 111011 111010; 110000 110001 110101 110100 111100 111101 111001 111000].

Example 4.2. Let $S:(Source) = 101111$ and $D:(Destination) = 010010$ be the two nodes selected from the label set of $CSNG(2,2)$. Mapping for this graph is obtained as in Fig. 6 using Algorithm 2. The method is to first determine the location of S and D on the map. The way to detect this is graph traversal. For this example, the location of S is $SI = \{7,6\}$ and the location of D is $DI = \{3,1\}$. It is understood from this locations to reach D from S , path will go through 9 nodes that is equal to the difference between the two locations as $|SI(1) - DI(1)| + |SI(2) - DI(2)| = 4 + 5 = 9$. After finding the locations of S and D, the path is formed by first moving vertically and then horizontally by Algorithm 3. The locations of the nodes on this route are

$$\{7,6\} \rightarrow \{6,6\} \rightarrow \{5,6\} \rightarrow \{4,6\} \rightarrow \{3,6\} \rightarrow \{3,5\} \rightarrow \{3,4\} \rightarrow \{3,3\} \rightarrow \{3,2\} \rightarrow \{3,1\}$$

where $SI = \{7,6\}$ and $DI = \{3,1\}$. So, the route is

$$101111 \rightarrow 111111 \rightarrow 111101 \rightarrow 011101 \rightarrow 011111 \rightarrow 011110 \rightarrow 010110 \rightarrow 010111 \rightarrow 010011 \rightarrow 010010$$

Example 4.3. The Example 4.3 (map algorithm) maps the labels for the graph $CSNG(k,m)$ in the $2D$ plane as Fig. 6. The algorithm is recursive and works in the same way as the labeling algorithm given in [17]. This algorithm returns an array of size such that the position of the labels in the $2D$ plane is equal to the number of vertices in the graph. Time complexity of the Algorithm 2 running time is $2^{k+1} - 1 + 2^{m+1} - 1$ as the calculation of the complexity of Algorithm 1. When $k > m$ the running time will be $\mathcal{O}(2^{k+1} - 1)$ and when $k < m$ the running time will be $\mathcal{O}(2^{m+1} - 1)$. So, there are two cases of the running time of Algorithm 2 as follows;

- *If k and m are different:* The running time of the Algorithm 2 is $\mathcal{O}(z)$ where $z = max(2^{k+1} - 1, 2^{m+1} - 1)$,
- *If k and m are equal:* The running time of the Algorithm 2 is $\mathcal{O}(z)$ where $z = 2^{k+2} - 2$.

The Algorithm 3 finds unicast routing for $CSNG(k,m)$ with the help of Algorithm 2. Time complexity of the Algorithm 3 is $\mathcal{O}(2^{k+m+2})$.

Algorithm 2: This algorithm is mapping labelling of nodes of $CSNG(k,m)$ using divide and conquer process.

```
Data: $k, m$
Result: Labelling of $CSNG(k,m)$
Function Map($k,m,i,j$):
    if $k>0$ then
        $M(1:2^k,:) = j||Map(k-1,m,0,j)$
        $M(2^k+1:2^{k+1},:) = !j||Map(k-1,m,0,!j)$
        return $M$
    end
    if $m>0$ then
        $M(:,1:2^m) = i||Map(k,m-1,0,j)$
        $M(:,2^m+1:2^{m+1}) = !i||Map(k,m-1,1,j)$
        return $M$
    end
    if $i==0$ and $j==0$ then
        return $M$=[00 01; 10 11]
    end
    if $i==1$ and $j==0$ then
        return $M$=[01 00; 11 10]
    end
    if $i==0$ and $j==1$ then
        return $M$=[10 11; 00 01]
    end
    if $i==1$ and $j==1$ then
        return $M$=[11 10; 01 00]
    end
End Function
```

Algorithm 3: This algorithm calculates unicast routing for $CSNG(k,m)$ (iterative process).

```
Data: $S$ = source, $D$ = destination, $M : Map, k, m,$
      $MSG$
Result: $MSG$ message is transmitted from $S$ to $D$
Function Route($S,D,M,k,m$):
    for $i = 0$ to $2^{k+1}-1$ do
        for $j = 0$ to $2^{m+1}-1$ do
            if $M[i,j] == S$ then
                $SI = [i,j]$
            end
            if $M[i,j] == D$ then
                $DI = [i,j]$
            end
        end
    end
    if $SI(1)! = D(1)$ then
        for $i = S(1)$ to $D(1)$ do
            $P = P \cup M(i, SI(2))$
        end
    end
    if $S2(1)! = D(2)$ then
        for $i = S(2)$ to $D(2)$ do
            $P = P \cup M(DI(1), i)$
        end
    end
End Function
for $i = 0$ to $L-1$ do
    $P[i+1] \overset{MSG}{\longleftarrow} P[i]$
end
```

The XY routing algorithm is a distributed routing algorithm used on $2D$ mesh networks [54]. This algorithm calculates the next node with using the current node address and the destination node address. Calculation of the next node takes constant time and it is done at each node on the path. So, the time complexity of the XY routing algorithm on $CSNG(k,m)$ is $\Theta(2^{k+m+2})$. Thus; Algorithm 3 and the XY routing algorithm have the same time complexity For Example 4.2, the path created by Algorithm 3 is shown in Fig. 7.

4.1.1. Parallelization

Through the utilization of parallel programming, it becomes possible to solve larger problems within a shorter time frame, thereby enhancing performance. Parallel programming method helps to provide the required acceleration and efficiency by dividing different parts of the problems into different processors for the solution of complex and large problems that have emerged with scientific developments. In parallel programming, the solving a problem is performed with the following steps [55];

- Identifying the tasks that can be done simultaneously.
- Mapping the tasks that are concurrent to the parallel processes.
- Distribution of the program's intermediate data input and output. Managing the use of data that is shared by processors.
- Bringing the processors into sync at various points throughout the parallel program execution.

The most basic preference reason for parallel programming is to prevent slowdowns in the computer by making the most appropriate use of memory by using multiple processors instead of a single processor. In this way, computers can perform calculations quickly and this provides an increase in performance on the computer.

Mapping of labels for $CSNG(k,m)$ can be perform in parallel to increase efficiency and decrease running time. At each iteration a map of subgraph $CSNG(k,m)$ can be created while each label is computed on a different processor. When the function calls itself, it divides the problem in half and when the problem size is one, it calculates the map for $CSNG(0,0)$. Then, the algorithm calculates a sub problem that is twice the size of the lower sub problem using the result of this lower sub problem.

There are two different cases for running time of the Algorithm 2 in parallel architecture depends on the address length of nodes and the number of processors. Here, address length for $CSNG(k,m)$ is $k+m+2$. For Algorithm 3 there is only one case for the run-time complexity.

Case 1. *(If the number of processor 2^n is greater than or equal to the number of nodes)* If Algorithm 2 is executed on nodes in parallel, the running time will be $k+m$ in this case. The reason for this is that when computing the labels, 2^i labels can be computed each time so that i can take values from 2 to $k+m$. For$CSNG(k,m)$, $CSNG(k-1,m)$ or $CSNG(k,m-1)$ is used to compute the map and this step takes place in parallel in $\mathcal{O}(1)$ time on different processors. When the map of the graph $CSNG(k,m)$ is computed from the base case $CSNG(0,0)$, it takes $\mathcal{O}(k+m)$ time. When $2^{k+m} \leqslant 2^n$ and $n \in N$, running time of Algorithm 2;

$T(n) = k+m.$

For Algorithm 3, the running time will be equal to the sending message through path. Finding the location of the source and destination nodes on the map takes $\mathcal{O}(1)$ time, at each node the source and destination node value is compared with the label value of that node and then the same ones are returned as row and column numbers of the graph, only the comparison time is used to calculate the running time of the parallel algorithm for location of

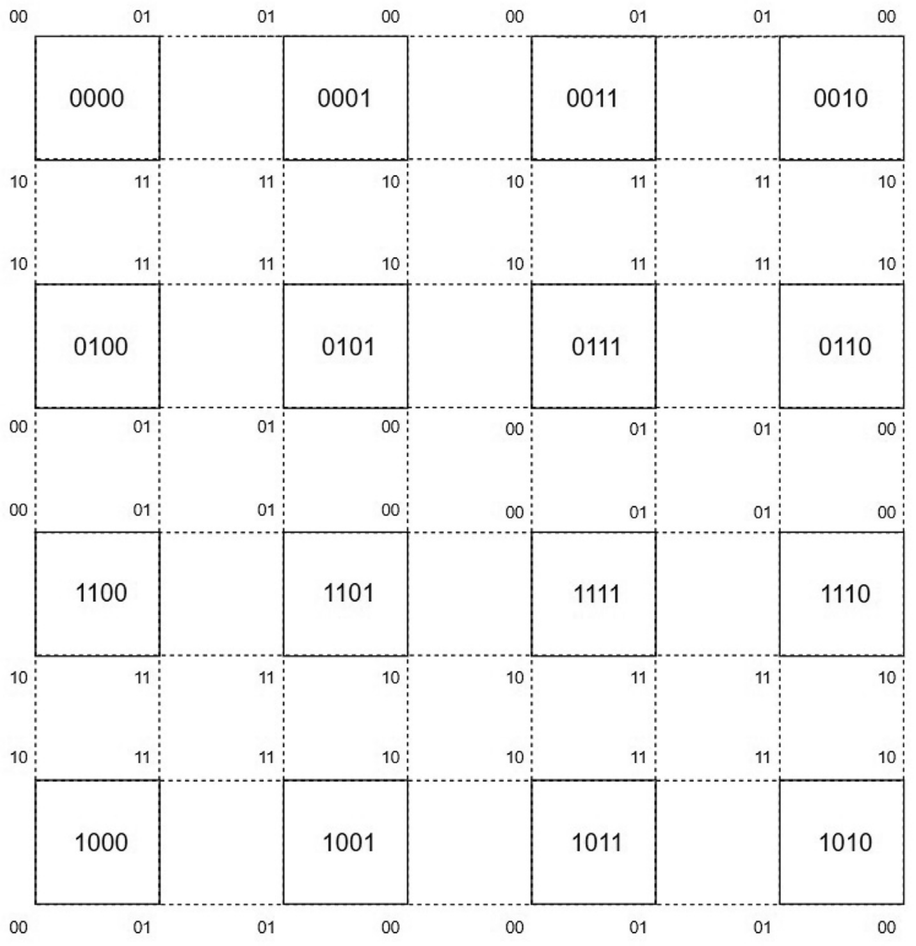

Fig. 6. Mapping process: labels of all the squares that make up $CSNG(2,2)$.

nodes. Once the location of the source and the destination node in the map is known, the number of vertices in a row and in a column of the map will be the running time of algorithm $2^{k+1}-1+2^{m+1}-1$ that is $\mathcal{O}(2^k+2^m)$ that is the message sending on 2*D* graph node by node in the worst case scenario. When $2^{k+m}\leqslant 2^n$ and $n\in N$, running time of the Algorithm 3;

$T(n)=\mathcal{O}(2^k+2^m)$.

Case 2. *(If the number of processor is less than* 2^{k+m+2}*)* The running time of the Algorithm 2 will be equal to $n+2^{addresslength-n}$ where the *addresslength* is $m+k+2$ and the number of processor is 2^n. It takes n times to compute the label of each node until the graph size is 2^n, but once the size is larger than 2^n, computation number increases proportionally to the power of 2. When $k+m>2^n$ and $n\in N$, running time of Algorithm 2;

$T(n)=\mathcal{O}(n+2^{addresslength-n})=\mathcal{O}(n+2^{k+m+2-n})$

The running time of Algorithm 3 is same as Case 1, which is $\mathcal{O}(2^k+2^m)$ that is the sum of a number of vertices in a column and a number of vertices in a row in a $CSNG(k,m)$ equals to message sending on 2*D* graph node by node in the worst case scenario. When $k+m>2^n$ and $n\in N$, running time of Algorithm 3;

$T(n)=\mathcal{O}(2^k+2^m)$

4.2. Broadcasting algorithms

This subsection will focus on broadcasting algorithms, utilizing hypercube's broadcasting algorithms.

Remark 4.1. Iterative version of one-to-all and all-to-all broadcasting algorithms for $CSNG(k,m)$ can be obtained similarly by writing $k+m+2$ instead of $2k+2$ in Algorithms 2 and 3 in [15]. Algorithm 4 and Algorithm 5 are the revised version of hypercube broadcast algorithms in [44] designed using divide and conquer approach. The costs for these two algorithms can be obtained by writing $k+m+2$ instead of *logp* the complexity in Algorithm 4.2 and Algorithm 4.7 in [44].

According to cost analysis in [44], total time for one to all broadcasting has same equation and is equal to *logp* times time cost of point to point simple message transfer. Assume t_s is the time required for start-up an t_w is the time sending one word from one node to other node. To send a message with size n from one node to all nodes, time complexity will be $T=(t_s+t_wn)(k+m+2)$. This algorithm has the same time complexity with the broadcast algorithm for mesh networks. For all-to-all broadcast algorithms, time complexity is similar with hyper-

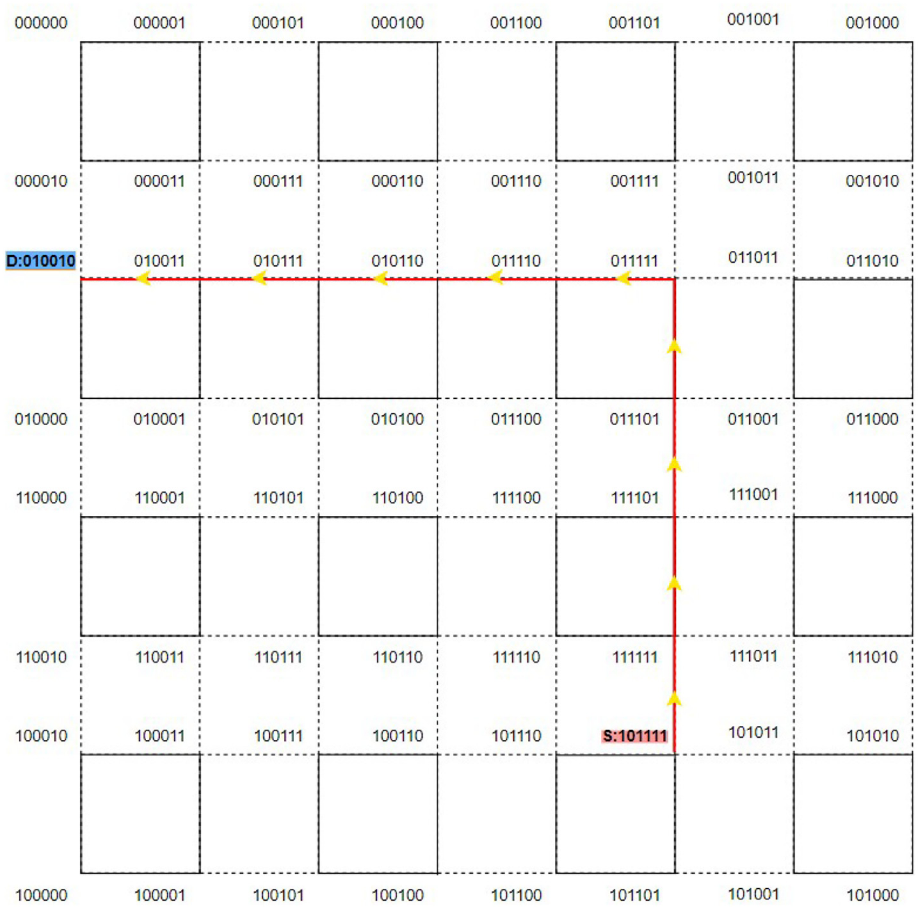

Fig. 7. Unicast routing for $CSNG(2,2)$.

cube and the equation for time complexity is $T = t_s(k+m+2) + t_w n(m+k+1)$. Time for all-to-all broadcasting algorithm for mesh network is equal to $T = 2t_s(\sqrt{p}$-1)+t_w $n(p-1)$ where p is the number of node in the network. Since $2t_s$ ($\sqrt{p}$-1) >$t_s logp$ for large number of p, all-to-all broadcast algorithm for $CSNG(k,m)$ has better performance than for mesh network.

Algorithm 4: This algorithm calculates one to all broadcasting for $CSNG(k,m)$ using divide and conquer approach.

```
   Data: $my\_label = 0,\ t = 0$
   Result: $MSG$ message is transmitted using all
           nodes
1  Function All_to_all($k, m, my\_label, my\_MSG, t$):
2      if $t < k+m+2$ then
3          $partner\_label = my\_label \bigoplus 2^t$
4          send my_MSG to $partner\_label$
5          receive $MSG$ from $partner\_label$
6          $my\_MSG = my\_MSG \cup MSG$
7          All_to_all($k, m, my\_label, my\_MSG, t+1$)
8      end
9  End Function
```

Algorithm 5: This algorithm calculates all to all broadcasting for $CSNG(k,m)$ using divide and conquer approach.

```
    Data: $my\_label = 0,\ t = k+m+1,$
          $mask = (2^{k+m+2}-1) \bigoplus 2^{m+k+1}$
    Result: $MSG$ message is transmitted from one
            node to all nodes
1   Function OnetoAll($k, m, my\_label, MSG, mask, t$):
2       if $t >= 0$ then
3           if $my\_label \cap mask == 0$ then
4               if $my\_label \cap 2^t == 0$ then
5                   $destination\_label = my\_label \bigoplus 2^t$
6                   send $MSG$ to $destination\_label$
7               end
8               else
9                   $source\_label = my\_label \bigoplus 2^t$
10                  receive $MSG$ from $source\_label$
11              end
12          end
13          mask = mask $\cap 2^{t-1}$
14          OnetoAll($k, m, my\_label, MSG, mask, t-1$)
15      end
16  End Function
```

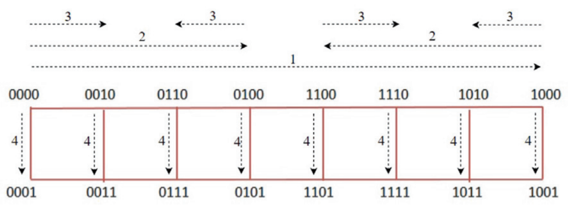

Fig. 8. All steps for one to all broadcasting for $CSNG(k, m)$.

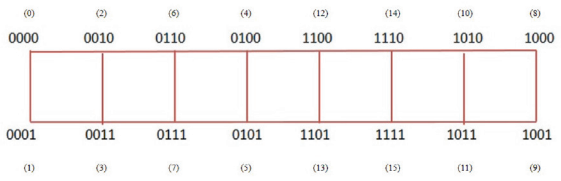

Fig. 9. Base case for all to all broadcasting for $CSNG(k, m)$.

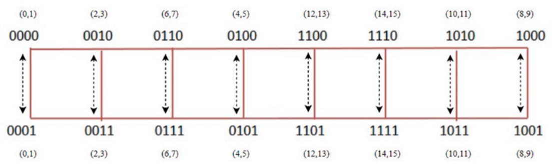

Fig. 10. First step for all to all broadcasting for $CSNG(k, m)$.

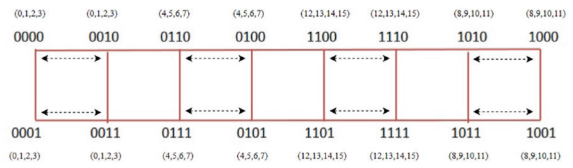

Fig. 11. Second step for all to all broadcasting for $CSNG(k, m)$.

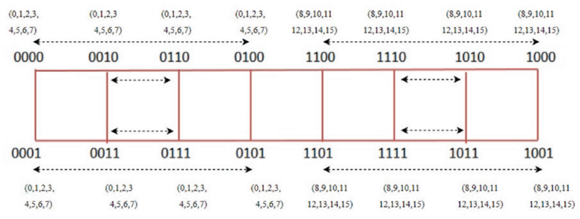

Fig. 12. Third step for all to all broadcasting for $CSNG(k, m)$.

Actually, since this mesh structure $CSNG(k, m)$ is constructed as a hypercube variant, the broadcasting algorithms for this mesh structure exhibit different behavior than the standard mesh algorithms (see Fig. 8–13).

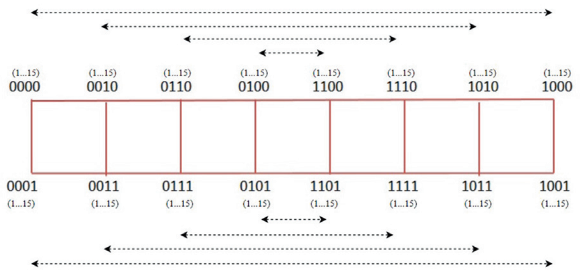

Fig. 13. Fourth step for all to all broadcasting for $CSNG(k, m)$.

5. Conclusions

This study focuses on reevaluating Connected Square Network Graphs ($CSNG(k, m)$), which is a subgraph of 2D-meshes. The key feature of $CSNG(k, m)$ is its resemblance to a hypercube variant. The primary objective of this study is to consider $CSNG(k, m)$ as a hypercube variant and to develop algorithms that offer solutions to significant problems. As a result, these algorithms obtained for this graph are similar to hypercube algorithms rather than classical mesh algorithms. . The first algorithm aims to find an alternative Hamiltonian path for $CSNG$ by using dynamic programming, with a runtime of $\mathcal{O}(2^{k+m+2})$. The second algorithm calculates the mapping of node labels for $CSNG(k, m)$ and runtime of it is $\mathcal{O}(2^{k+m+2})$. Additionally, Algorithm 3 presents an unicast routing algorithm with time complexity of $\mathcal{O}(max(2^{k+1} - 1, 2^{m+1} - 1))$. The utilization of parallel and distributed architecture enhances performance at runtime, depending on the number of processors. The most general form of the runtime calculations for the parallel Algorithm 2 equals to $\mathcal{O}(n + 2^{k+m+2-n})$ and running time of parallel Algorithm 3 is $\mathcal{O}(2^k + 2^m)$ where address length of $CSNG(k, m)$ is $k + m + 2$. Broadcasting algorithms for $CSNG$ can be easily derived using similar strategy as the broadcasting algorithms used for hypercube. For a specific case of 2D meshes, a different approach is presented using well-known hypercube algorithms.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The authors are very grateful to the referees for their valuable suggestions, which helped to improve the paper significantly.

References

[1] J.P. Hayes, T.N. Mudge, Q.F. Stout, Architecture of a hypercube supercomputer, in, Proc. ICPP (1986) 653–660.
[2] J. Nieminen, M. Peltola, P. Ruotsalainen, J. Mieminen, On graphs like hypercubes, Tsukuba J. Math. 32 (1) (2008) 37–48. https://doi.org/10.21099/tkbjm/1496165191.
[3] G.B. Chae, E.M. Palmer, R.W. Robinson, Counting labeled general cubic graphs, Discrete Math. 307 (2007) 2979–2992, https://doi.org/10.1016/j.disc.2007.03.011.
[4] F. Harary, J.P. Hayes, H.-J. Wu, A survey of the theory of hypercube graphs, Comput. Math. Appl. 15 (4) (1988) 277–289.
[5] W. Mao, D.M. Nicol, On k-ary n-cubes: theory and applications, Discrete Appl. Math. 129 (2003) 171–193, https://doi.org/10.1016/S0166-218X(02)00238-X.
[6] A. El-Amawy, S. Latifi, Properties and performance of folded hypercubes, IEEE Trans. Parallel Distrib. Syst. 2 (1) (1991) 31–42.
[7] H.Y. Chang, R.J. Chen, Incrementally extensible folded hypercube graphs, J. Inf. Sci. Eng. 16 (2) (2000) 291–300, https://doi.org/10.1109/ICPADS.1998.741133.

[8] Q. Dong, X. Yang, J. Zhao, Y.Y. Tang, Embedding a family of disjoint 3D meshes into a crossed cube, Inf. Sci. 178 (2008) 2396–2405, https://doi.org/10.1016/j.ins.2007.12.010.
[9] K. Efe, The crossed cube architecture for parallel computation, IEEE Trans. Parallel Distrib. Syst. 40 (1991) 1312–1316.
[10] C.J. Lai, C.H. Tsai, H.C. Hsu, T.K. Li, A dynamic programming algorithm for simulation of a multi-dimensional torus in a crossed cube, Inf. Sci. 180 (2010) 5090–5100, https://doi.org/10.1016/j.ins.2010.08.029.
[11] M. Abd-El-Barr, T.F. Soman, Topological properties of hierarchical interconnection networks a review and comparison, J. Electr Comput. Eng. (2011), https://doi.org/10.1155/2011/189434.
[12] K. Ghose, K.R. Desai, Hierarchical cubic networks, IEEE Trans. Parallel Distrib. Syst. 6 (1995) 427–435.
[13] A. Karci, Hierarchical extended fibonacci cubes, Iran. J. Sci. Technol. Trans. B Eng. 29 (2005) 117–125.
[14] A. Karci, Hierarchic graphs based on the fibonacci numbers, Istanbul Univ, J. Electr Electron. Eng. 7 (1) (2007) 345–365.
[15] A. Karci, B. Selcuk, A new hypercube variant : Fractal Cubic Network Graph, Eng. Sci. Technol., Int. J. 18 (1) (2014) 32–41, https://doi.org/10.1016/j.jestch.2014.09.004.
[16] B. Selcuk, A. Karci, Connected Cubic Network Graph, Eng. Sci. Technol., Int. J. 20 (3) (2017) 934–943, https://doi.org/10.1016/j.jestch.2017.04.005.
[17] B. Selçuk, Connected Square Network Graphs, Universal J. Math. Appl. 5 (2) (2022) 57–63. https://doi.org/10.32323/ujma.1058116.
[18] B. Selçuk, A.N.A. Tankül, A New Hypercube-like Graph, Turkish Journal of Mathematics-Studies on Scientific Developments in Geometry, Algebra, and Applied Mathematics , (pp. 85), Ankara, Türkiye, (February 2022)
[19] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms , Second Edition, vol. 7. The MIT Press, 2001.
[20] X. Wang, Z.G. Mou, A divide-and-conquer method of solving tridiagonal systems on hypercube massively parallel computers, in: Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, Dallas, TX, 1991, pp. 810–817, https://doi.org/10.1109/SPDP.1991.218237.
[21] E.W. Mayr, R. Werchner, Divide-and-conquer algorithms on the hypercube, Theoret. Comput. Sci. 162 (2) (1996) 283–296, https://doi.org/10.1016/0304-3975(96)00033-3.
[22] S. Lor, H. Shen, P. Maheshwari, Divide-and-conquer mapping of parallel programs onto hypercube computers, J. Syst. Architect. 43 (6–7) (1997) 373–390, https://doi.org/10.1016/S1383-7621(96)00052-5.
[23] A. Karci, Generalized parallel divide and conquer on 3D mesh and torus, J. Syst. Architect. 51 (5) (2005) 281–295, https://doi.org/10.1016/j.sysarc.2004.06.004.
[24] M. Valero-García, A. González, L. Díaz de Cerio, D. royo Valles, Divide-and-Conquer Algorithms on Two-Dimensional Meshes, 1998. https://doi.org/1051-1056. 10.1007/BFb0057966.
[25] M. Manzoor, R.N. Mir, N. Hakim, PAAD (Partially adaptive and deterministic routing): A deadlock free congestion aware hybrid routing for 2D mesh network-on-chips, Microprocessors Microsystems, Volume 92 104551 (2022), https://doi.org/10.1016/j.micpro.2022.104551.
[26] R. Panja, S.S. Vadhiyar, HyPar: A divide-and-conquer model for hybrid CPU–GPU graph processing, J. Parallel Distributed Computing 132 (2019) 8–20, https://doi.org/10.1016/j.jpdc.2019.05.014.
[27] P.K. Tripathy, R.K. Dash, C.R. Tripathy, A dynamic programming approach for layout optimization of interconnection networks, Eng. Sci. Technol., Int. J. 18 (3) (2015) 374–384, https://doi.org/10.1016/j.jestch.2015.01.003.
[28] J. Crichigno, J. Khoury, M.Y. Wu and W. Shu, A Dynamic Programming Approach for Routing in Wireless Mesh Networks, in: IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference, New Orleans, LA, USA, 2008, pp. 1-5, https://doi.org/10.1109/GLOCOM.2008.ECP.108.
[29] Y. Hou, C.M. Wang, C.Y. Ku, L.H. Hsu, Optimal processor mapping for linear-complement communication on hypercubes, IEEE Trans. Parallel Distrib. Syst. 12 (5) (May 2001) 514–527, https://doi.org/10.1109/71.926171.
[30] S.A. Strate, R.L. Wainwright, Parallelization of the dynamic programming algorithm for the matrix chain product on a hypercube, in: Proceedings of the 1990 Symposium on Applied Computing, USA, 1990, pp. 78–84, https://doi.org/10.1109/SOAC.1990.82144.
[31] A. Goldman and D. Trystram, An efficient parallel algorithm for solving the knapsack problem on the hypercube, Proceedings 11th International Parallel Processing Symposium, Genva, Switzerland, 1997, pp. 608-615, https://doi.org/10.1109/IPPS.1997.580964.
[32] G. Reinelt, The Traveling Salesman: Computational Solutions for TSP Applications, Springer-Verlag, 1994.
[33] M. Manfrin, M. Birattari, T. Stützle, M. Dorigo, Parallel Ant Colony Optimization for the Traveling Salesman Problem, Lect. Notes Comput. Sci. (2006) 224–234, https://doi.org/10.1007/11839088_20.
[34] A. Shaheen, A. Sleit, S. AlSharaeh, Chemical Reaction Optimization for Traveling Salesman ProblemOver a Hypercube Interconnection, Network (2018) 432–442, https://doi.org/10.1007/978-3-319-91192-2_43.
[35] A. Shaheen, A. Sleit, and S. Al-Sharaeh, Travelling Salesman Problem Solution Based-on Grey Wolf Algorithm over Hypercube Interconnection Network, Modern Applied Science, vol. 12, no. 8, 2018. doi: 10.5539/mas.v12n8p142.
[36] Al. Paler, A. Zulehner, R. Wille, NISQ circuit compilation is the travelling salesman problem on a torus. Quantum, Science Technol. (2021), https://doi.org/10.1088/2058-9565/abe665.
[37] N.S. Benni, S.S. Manvi, Channel Allocation Scheme to Mitigate Interference in 5G Backhaul Wireless Mesh Networks, in: 2022 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI), Chennai, India, 2022, pp. 1–5, https://doi.org/10.1109/ICDSAAI55433.2022.10028841.
[38] R.J. Wilson, Introduction to Graph Theory, 4th ed., Addison Wesley Longman Limited, Harlow, 1996.
[39] N. Phisutthangkoon, J. Werapun, Shortest-path routing for optimal all-to-all personalized-exchange embedding on hierarchical hypercube networks, J. Parallel Distributed Comput. 150 (2021) 139–154, https://doi.org/10.1016/j.jpdc.2021.01.004.
[40] T. Fu, C. Li, L. Wu, L. Zou, A specific type of irregular ring-and-hub network structure and the average shortest distance of its rings, Heliyon 8 (11) (2022) , https://doi.org/10.1016/j.heliyon.2022.e11470 e11470.
[41] A.N. Onaizah, Y. Xia, K. Hussain, A. Mohamed, Optimized shortest path algorithm for on-chip board processor in large scale networks, Optik 271 (2022) , https://doi.org/10.1016/j.ijleo.2022.170151 170151.
[42] C.N. Lai, Constructing all shortest node-disjoint paths in torus networks, J. Parallel Distributed Comput. 75 (2015) 123–132, https://doi.org/10.1016/j.jpdc.2014.09.004.
[43] C.M. Kozierok, The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference. No Starch Press, 1st edition, 2005. ISBN-13: 978-1593270476
[44] V. Kumar, A. Grama, A. Gupta, G. Karypis Introduction to Parallel Computing: Design and Analysis of Algorithms The Benjamin/Cummings Publishing Company (1994).
[45] A. Maurer, S. Tixeuil, Byzantine broadcast with fixed disjoint paths, J. Parallel Distributed Comput. 74 (11) (2014) 3153–3160, https://doi.org/10.1016/j.jpdc.2014.07.010.
[46] A. Samuylov, D. Moltchanov, R. Kovalchukov, A. Gaydamaka, A. Pyattaev, Y. Koucheryavy, GAR: Gradient assisted routing for topology self-organization in dynamic mesh networks, Comput. Commun. 190 (2022) 10–23, https://doi.org/10.1016/j.comcom.2022.03.023.
[47] G. De Marco, U. Vaccaro, Broadcasting in hypercubes and star graphs with dynamic faults, Information Processing Letters 66 (6) (1998) 321–326, https://doi.org/10.1016/S0020-0190(98)00074-X.
[48] H.-P. Ye, W.-J. Xiao, J.-Y. Wu, Broadcasting on the BSN-Hypercube Network, in: 2009 Second International Conference on Information and Computing Science, Manchester, UK, 2009, pp. 167–170, https://doi.org/10.1109/ICIC.2009.49.
[49] N. Phisutthangkoon, J. Werapun, Optimal ATAPE task scheduling on reconfigurable and partitionable hierarchical hypercube networks, Parallel Comput. 111 (2022) , https://doi.org/10.1016/j.parco.2022.102923 102923.
[50] A.Y. Al-Dubai, M. Ould-Khaoua, On the design of scalable pipelined broadcasting for mesh networks, Proc. - Int, Symp. High Perform. Comput. Syst. Appl. vol. 2002 (2002) 98–105, https://doi.org/10.1109/HPCSA.2002.1019140.
[51] A. Itai, C. Papadimitriou, J. Szwarcfiter, Hamilton paths in grid graphs, SIAM J. Comput. 11 (4) (1982) 676–686.
[52] S.D. Chen, H. Shen, R. Topor, An efficient algorithm for constructing Hamiltonian paths in meshes, Parallel Comput. 28 (9) (2002) 1293–1305, https://doi.org/10.1016/S0167-8191(02)00135-7.
[53] F. Keshavarz-Kohjerdi, A. Bagheri, A. Asgharian-Sardroud, A linear-time algorithm for the longest path problem in rectangular grid graphs, Discrete Appl. Math. 160 (3) (2012) 210–217, https://doi.org/10.1016/j.dam.2011.08.010.
[54] W. Zhang, L. Hou, J. Wang, S. Geng, W. Wu, Comparison Research between XY and Odd-Even Routing Algorithm of a 2-Dimension 3X3 Mesh Topology Network-on-Chip, in: 2009 WRI Global Congress on Intelligent Systems, 2009, https://doi.org/10.1109/gcis.2009.110.
[55] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Addison Wesley, 2003.