Electronic Journal of Graph Theory and Applications **9** (2) (2021), 507–525

Electronic Journal of Graph Theory and Applications

Embedding complete multipartite graphs into Cartesian product of paths and cycles

R. Sundara Rajan[a], A. Arul Shantrinal[a], T.M. Rajalaxmi[b], Jianxi Fan[c], Weibei Fan[d]

[a]*Department of Mathematics, Hindustan Institute of Technology and Science, Chennai, India, 603 103*
[b]*Department of Mathematics, Sri Sivasubramaniya Nadar College of Engineering, Chennai, India, 603 110*
[c]*School of Computer Science and Technology, Soochow University, Suzhou 215006, China*
[d]*School of Computer Science & Technology, Nanjing University of Posts and Telecommunications, Jiangsu 210049, China*

vprsundar@gmail.com, shandrinashan@gmail.com (Corresponding author), laxmi.raji18@gmail.com, jxfan@suda.edu.cn, fanweibei@163.com

Abstract

Graph embedding is a powerful method in parallel computing that maps a guest network G into a host network H. The performance of an embedding can be evaluated by certain parameters, such as the dilation, the edge congestion and the wirelength. In this manuscript, we obtain the wirelength (exact and minimum) of embedding complete multi-partite graphs into Cartesian product of paths and/or cycles, which include n-cube, n-dimensional mesh (grid), n-dimensional cylinder and n-dimensional torus, etc., as the subfamilies.

Keywords: embedding, edge congestion, wirelength, complete multi-partite graphs, Cartesian product of graphs
Mathematics Subject Classification: 05C60, 05C85
DOI: 10.5614/ejgta.2021.9.2.21

1. Introduction and Preliminaries

Given two graphs G (guest) and H (host), an embedding from G to H is an injective mapping $f : V(G) \to V(H)$ and associating a path $P_f(e)$ in H for each edge e of G. We, now define the edge congestion $EC(G, H)$ and the wirelength $WL(G, H)$ [4] as follows:

- $EC(G, H) = \min\limits_{f:G\to H} \max\limits_{e=xy\in E(H)} EC_f(e)$
- $WL(G, H) = \min\limits_{f:G\to H} \sum\limits_{e=xy\in E(G)} \text{dist}_H(f(x), f(y)) = \min\limits_{f:G\to H} \sum\limits_{e=xy\in E(H)} EC_f(e)$

Received: 27 September 2019, Revised: 19 June 2021, Accepted: 14 July 2021.

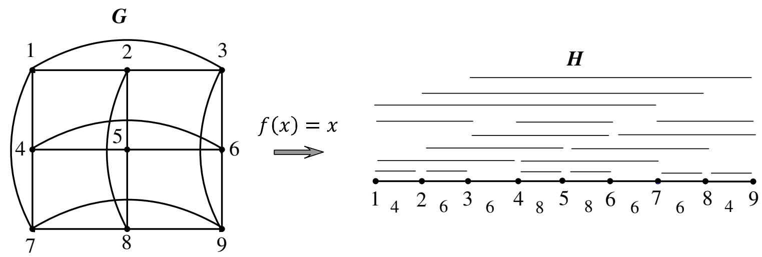

Figure 1. An embedding f from torus G into a path H with $EC_f(G,H) = 8$ and $WL_f(G,H) = 48$.

where $\text{dist}_H(f(x), f(y))$ is a distance (need not be a shortest distance) between $f(x)$ and $f(y)$ in H and $EC_f(e)$ denote the number of edges e' of G such that $e = xy$ is in the path $P_f(e')$ (need not be a shortest path) between $f(x)$ and $f(y)$ in H. Further, $EC_f(S) = \sum_{e \in S} EC_f(e)$, where $S \subseteq E(H)$.

For example, the edge congestion and the wirelength of an embedding $f : C_3 \square C_3 \to P_9$ is given in Fig. 1. It is easy to observe that, the above two parameters are different. But, for any embedding g, the sum of the edge congestion (called edge congestion sum) and the wirelength are all equal. Mathematically, we have the following equality.

$$\sum_{e=xy \in E(H)} EC_g(e) = WL_g(G,H).$$

In this manuscript, we will use the edge congestion sum to estimate the wirelength. Further, if $n \geq 1$, then the set $\{1, 2, \ldots, n\}$ will be denoted by $[n]$.

For a subgraph M of G of order n,

- $I_G(M) = \{uv \in E \mid u, v \in M\}, \;\; I_G(k) = \max_{M \subseteq V(G),\, |M|=k} |I_G(M)|$
- $\theta_G(M) = \{uv \in E \mid u \in M, v \notin M\}, \;\; \theta_G(k) = \min_{M \subseteq V(G),\, |M|=k} |\theta_G(M)|$

The *maximum subgraph problem* (MSP) for a given k, $k \in [n]$ is a problem of computing a subset M of $V(G)$ such that $|M| = k$ and $|I_G(M)| = I_G(k)$. Further, the subsets M are called the *optimal set* [17, 5, 19]. Similarly, we define the *minimum cut problem* (MCP) for a given k, $k \in [n]$ is a problem of computing a subset M of $V(G)$ such that $|M| = k$ and $|\theta_G(M)| = \theta_G(k)$. For a regular graph, say r, we have $2I_G(k) + \theta_G(k) = rk$, $k \in [n]$ [5].

The following lemmas are efficient techniques to find the exact wirelength using MSP and MCP.

Lemma 1.1. [27] *Let $f : G \to H$ be an embedding with $|V(G)| = |V(H)|$. Let $S \subseteq E(H)$ be such that $E(H) \setminus S$ has exactly two subgraphs H_1 and H_2, and let $G_1 = G[f^{-1}(V(H_1))]$ and $G_2 = G[f^{-1}(V(H_2))]$. Furthermore, let S satisfy the following conditions:*

1. *For every $uv \in E(G_i), i \in [2]$,* the path *$P_f(uv)$ has no edges in S.*
2. *For every $uv \in E(G)$, $u \in V(G_1)$, $v \in V(G_2)$, the path $P_f(uv)$ has exactly one edge in S.*

3. *$V(G_1)$ and $V(G_2)$ are optimal sets.*

Then $EC_f(S)$ is minimum over all embeddings $f : G \to H$ and $EC_f(S) = \sum_{v \in V(G_1)} deg_G(v) - 2|E(G_1)| = \sum_{v \in V(G_2)} deg_G(v) - 2|E(G_2)|$, where $deg_G(v)$ is the degree of a vertex v in G.

Remark 1.1. For a regular graph G, it is easy to check that, $V(G_2)$ is optimal if and only if $V(G_1)$ is optimal and vice-versa [25].

Lemma 1.2. [27] *Let $f : G \to H$. If $\{P_1, \ldots, P_t\}$ is a partition of $E(H)$, where each part P_i is an edge cut that satisfies the conditions of Lemma 1.1, then*

$$WL_f(G, H) = \sum_{i=1}^{t} EC_f(P_i).$$

Moreover, $WL(G, H) = WL_f(G, H)$.

The multipartite graph is one in all the foremost in style convertible and economical topological structures of interconnection networks. The multipartite has several wonderful options and its one in all the most effective topological structure of parallel processing and computing systems. In parallel computing, a large process is often decomposed into a collection of little sub processes which will execute in parallel with communications among these sub processes. Due to these communication relations among these sub processes the multipartite graph can be applied for avoiding conflicts in the network as well as multipartite networks helps to identify the errors occurring areas in easy way. A complete p-partite graph $G = K_{n_1,\ldots,n_p}$ is a graph that contains p independent sets containing n_i, $i \in [p]$, vertices, and all possible edges between vertices from different parts.

The Cartesian product technique is a very powerful technique for create a huger graph from given little graphs and it is very important technique for planning large-scale interconnection networks [45, 38]. Especially, the n-dimensional grid (cylinder and torus) structure of interconnection networks offer a really powerful communication pattern to execute a lot of algorithms in many parallel computing systems [45], which helps to arrange the interconnection network into sequence of sub processors (layers) in uniform distribution manner for transmits the data's in faster way without delay in sending the data packets (messages). Mathematically, we now defined the Cartesian product of graphs as follows:

Definition 1.1. [20] *The Cartesian product $G \Box H$ of (not necessarily connected) graphs G and H is the graph with the vertex set $V(G) \times V(G)$, vertices (u, v) and (u', v') being adjacent if either $u = u'$ and $vv' \in E(H)$, or $v = v'$ and $uu' \in E(G)$. If $G_1, G_2, \ldots, G_m$ are graphs of order $n_1, n_2, \ldots, n_m$ respectively, then the Cartesian product of m factors $G_1 \Box G_2 \Box \cdots \Box G_m$ is denoted by $\bigotimes_{i=1}^{m} G_i$.*

Remark 1.2. The graph $\bigotimes_{i=1}^{n} G_i$ is said to be an n-dimensional grid or torus or cylinder if all n factors are paths or cycles or any one of the factor is path and the remaining $(n-1)$ factors are cycles, respectively.

The graph embedding problem has been well-studied by many authors with a different networks [1,5,6,10,11–45], and to our knowledge, almost all graphs considered as a host graph is a unique family (for example: path P_n, cycle C_n, grid $P_n \square P_m$, cylinder $P_n \square C_m$, torus $C_n \square C_m$, hypercube Q_r and so on). In this paper, we overcome this by taking Cartesian product of paths and/or cycles as a host graph. Moreover, we obtain the wirelength of embedding complete 2^p-partite graphs $K_{2^{r-p},2^{r-p},\ldots,2^{r-p}}$ into the Cartesian product of $n \geq 3$ factors of respective order 2^{r_i}, $i \in [n]$, where $r_1 \leq r_2 \leq \cdots \leq r_n$, $r_1 + r_2 + \cdots + r_n = r$ and each factor is a path or a cycle, $r \geq 3, 1 \leq p < r$.

Lemma 1.3. [30] If G is a complete p-partite graph $K_{r,r,\ldots,r}$ of order pr, $p, r \geq 2$, then

$$I_G(k) = \begin{cases} \frac{k(k-1)}{2}, & k \leq p-1, \\ \frac{q^2p(p-1)}{2}, & l = qp, 1 \leq q \leq r. \\ \frac{(q-1)^2p(p-1)}{2} + j(q-1)(p-1) + \frac{j(j-1)}{2}, & l = (q-1)p + j, 1 \leq j \leq p-1, \\ & 2 \leq q \leq r. \end{cases}$$

2. Main Results

In this section we give an algorithm that computes the minimum wirelength of embedding complete 2^p-partite graphs $K_{2^{r-p},2^{r-p},\ldots,2^{r-p}}$ into Cartesian product of n factors are paths or cycles or any one of the factor is path and the remaining $(n-1)$ factors are cycles.

We start with an auxiliary algorithm that accordingly labels the vertices of the complete 2^p-partite graph $K_{2^{r-p},2^{r-p},\ldots,2^{r-p}}$. We thus have 2^r vertices partitioned into $l = 2^p$ parts. Initially, all the vertices are unlabeled. Then label the first vertex in each partition (upto l) by increment of 1 using clockwise direction. Now, label the second vertex in the first partition as $l+1$. Now, label the second vertex in the remaining each partition by increment of 1. Continue this process until all the 2^r vertices are labeled. The formal algorithm is given below as Algorithm 1.

Algorithm 1:

Input: $N = 2^r$ (Total number of elements)
$p \geq 1$, where 2^{r-p} represents the number of elements in the each partite
Output: Labeling of complete 2^p-partite graph $K_{2^{r-p},2^{r-p},\ldots,2^{r-p}}$

Step 1. Initialize (z, x), z represent the partite number and y represent the vertex position of the individual partite considered in the loop

Step 2. Initialize $j = 1$

Step 3. Start the below loop for the first partite

Step 4. for ($x \leftarrow 1$ to 2^{r-p}, increment x by 1)

Step 4.1. for ($z \leftarrow 1$ to 2^p, increment z by 1)

Step 4.1.1. print $(z, x) = j$
Step 4.1.2. Increment j value by 1

Step 5. Print the labeling of complete 2^p-partite graphs
Step 6. Repeat Step 4 upto 2^p-partite
Step 7. Repeat until 2^{r-p} vertices are labeled in each partite.

Lemma 2.1. *If $r, n \geq 3$ and $p \geq 1$, then Algorithm 1 labels the vertices of the complete 2^p-partite graph $K_{2^{r-p},2^{r-p},\ldots,2^{r-p}}$ with different integers from $[2^r]$.*

Proof. The graph $K_{2^{r-p},2^{r-p},\ldots,2^{r-p}}$ has 2^r vertices partitioned into $l = 2^p$ independent parts. Algorithm proceeds as described before. Specifically, Step 4.1.1 labels the first vertex of the first partition as 1, and continues the same process upto l^{th} partition by increment of 1. The second vertex of the first partition is labeled with $l + 1$ and do the numbering in clockwise direction. Repeat the same process (Step 4) until we reach the label 2^r. Hence Algorithm 1 labels all the vertices of the complete 2^p-partite graph uniquely from 1 to 2^r. This completes the proof of the lemma. □

I

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

II

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

III

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

IV

4	8	12	16
20	24	28	32
36	40	44	48
52	56	60	64

Figure 2. The complete 4-partite graph $K_{16,16,16,16}$.

To illustrate Algorithm 1, consider the complete 4-partite graph $K_{16,16,16,16}$ as illustrated in Fig. 2. By Algorithm 1, we have $N = 64$, $r = 6$ and $p = 2$. Initialize, $j = 1$ for $z = 1$ and $x = 1$, $z \in [16]$, $x \in [16]$, $(1, 1) = 1$ (i.e, $x \in [2^{r-p}]$, $z \in [2^p]$, $(z, x) = j$) and hence label the first vertex of the first partite as 1. Next increment j by 1. For $j = 2$, we have $z = 2$ and $x = 1$, $(1, 2) = 2$, Now label the first vertex of the second partite as 2 and so. Now go to Step 4, repeat the same process until we reach the label of the last (64^{th}) vertex and the algorithm ends.

An implementation of Algorithm 1 in python and two of its outputs are listed in Annexure I. We continue with an auxiliary algorithm that labels the Cartesian product of $n \geq 3$ factors of respective order 2^{r_i}, $i \in [n]$, where $r_1 + r_2 + \cdots + r_n = r$, $r_1 \leq r_2 \leq \cdots \leq r_n$ and each factor is a path or a cycle, $r \geq 3, 1 \leq p < r$.

Algorithm 2:

Input: The dimension $n \geq 3$ and the value of $r_1, r_2, \ldots, r_n$

Output: Labeling of Cartesian product of graphs $\bigotimes_{i=1}^{n} G_i$, where G_i's are either a path or a cycle

Step 1. Initialize R, M, L, and (x, y), where R represents the number of Cartesian product of $(n-1)^{th}$ dimension graph, M represents the number of two dimensional Cartesian product graph in the $(n-1)^{th}$ dimension, $L = 2^r$, $r = r_1 + r_2 + \ldots + r_n$ (i.e, total number of vertices), and in (x, y), x represents the column and y represents the row

Step 2. Initialize variables $j = 1$, $P = 1$, $y = 1$, $Q = 2^{r_1}$, $K = 2^{r_2}$

Step 3. Set $R = 1$

Step 4. for ($Z \leftarrow 1$ to M, increment Z by 1) // If $r_3 = r_n$, then $Z = 1$

- **Step 4.1.** for ($x \leftarrow 1$ to K, increment x by 1)
- **Step 4.2.** $Q = 2^{r_1}$
- **Step 4.3.** for ($y \leftarrow P$ to Q, increment y by 1)
  - **Step 4.3.1.** print $(x, y) = j$
  - **Step 4.3.2.** if $\frac{j}{2^{r_1} \times 2^{r_2} \times M \times R} = 1$, then $P = y$
    else
  - **Step 4.3.3** if $(y = 2^{r_1})$, then $y = 0$ and $Q = P - 1$
  - **Step 4.3.4** $j = j + 1$
- **Step 4.4.** End for
- **Step 4.5.** End for

Step 5. End for

Step 6. $R = R + 1$

Step 7. Repeat Step 4 for 2^{r_n} copies of $(n-1)$ dimensional graph

Step 8. Print the labeling of the Cartesian product of paths and cycles.

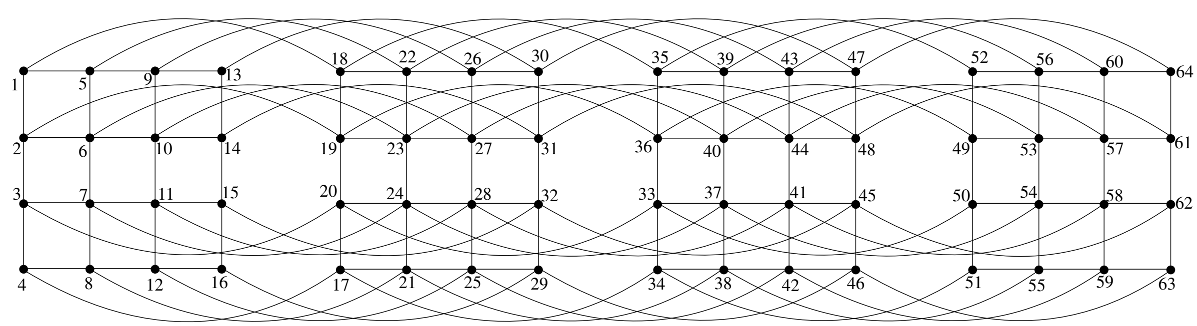

Figure 3. The Cartesian product of path graphs $P_4 \square P_4 \square P_4$.

Lemma 2.2. *If $r \geq 3$ and $i \in n$, then Algorithm 2 labels the Cartesian product of $n \geq 3$ factors of respective order 2^{r_i}, where $r_1 \leq r_2 \leq \cdots \leq r_n$, $r_1 + r_2 + \cdots + r_n = r$, $1 \leq p < r$ and each factor is a path or a cycle.*

Proof. The graph has 2^r vertices and dimension $n \geq 3$. Algorithm 2 proceeds as described before. We initialize the first two a_1 and a_2 parameter as the two dimensional graph. The a_3 parameter takes input which produces a_3 copies of the base dimension $(a_1 \times a_2)$. Similarly, the a_n parameter takes input which produces a_n copies of the base dimension $(a_1 \times a_2 \cdots \times a_{n-1})$. First label the $(n-1)$ dimensional graph. Then, update the row position and start the labeling in the second copy of $(n-1)$ dimensional graph. Repeat the same process (Step 4) until we reach the label 2^r. Hence Algorithm 2 labels the all the vertices of the Cartesian product of paths and cycles uniquely from 1 to 2^r. This completes the proof of the lemma. □

To illustrate Algorithm 2, we consider the graph $P_4 \square P_4 \square P_4$ as illustrated in Fig. 3. In Algorithm 2, we have $Q = 4$ and $K = 4$. Initialize, $R = 1$, $j = 1$, $P = 1$ for $Z = 1$, $x \in [4]$, $y \in [4]$, $(x, y) = (1, 1) = 1$ and hence label the first vertex of the 2-dimensional grid as 1. Then, go to Step 4.3.2, the condition $\frac{j}{2^{r_1} \times 2^{r_2} \times M \times R} = \frac{1}{4 \times 4 \times 1 \times 1} \neq 1$ fails. Next go to Step 4.3.3, the condition $y = 2^{r_1} \Rightarrow 1 \neq 4$ fails. Now increment j by 1. Go to Step 4.3, we have $j = 2$, $x = 1$ and $y = 2$, $(1, 2) = 2$. Now, label the first vertex of the second row as 2 and so, for $j = 5$ we have $x = 1$ and $y = 5$, the condition $y \nleq Q = 5 \nleq 4$ fails. Thus, go to Step 4.1, we have $x = 2$ and $y = 1$, $(2, 1) = 5$ hence label the second vertex of the first row as 5. Continue this process until we reach all vertices of the two dimensional grid. Next go to Step 4.3.2, we have $(4, 4) = 16$, so the condition $\frac{j}{2^{r_1} \times 2^{r_2} \times M \times R} = \frac{16}{4 \times 4 \times 1 \times 1} = 1$ satisfied. So, take the increment of R by 1. Now, $P = y$ (i.e, $P = 4$), so $(x, y) = (1, 4) = 17$, label the first vertex of the last row as 17. Go to Step 4.3.2, the condition fails. Now come to the else part, we have $4 = 4$ $(y = 2^{r_1})$. So, $y = 0$, $Q = P - 1 = 4 - 1 = 3$. Now increment j by 1. Do the same process until we reach the label of the last $(i.e., 64^{th})$ vertex.

The Python program and the corresponding implementation of Algorithm 2 are given in Annexure II.

We, now ready to prove the main result.

Theorem 2.1. *Let G be the complete 2^p-partite graphs $K_{2^{r-p}, 2^{r-p}, \ldots, 2^{r-p}}$ and H be the Cartesian product of $n \geq 3$ factors of respective order 2^{r_i}, $i \in [n]$, where $r_1 \leq r_2 \leq \cdots \leq r_n$, $r_1 + r_2 + \cdots + r_n = r$ and each factor is a path or a cycle, $r \geq 3, 1 \leq p < r$. Let $s \geq 0$ factors of H are paths and the remaining $(n - s)$ factors are cycles. Then the embedding f of G into H given by $f(x) = x$ with minimum wirelength and is given by,*

$$\begin{aligned} WL(G, H) &= \frac{2^{2r-p}(2^p - 1)}{6} \left[(2^{r_1} + 2^{r_2} + \cdots + 2^{r_s}) - \left(\frac{1}{2^{r_1}} + \frac{1}{2^{r_2}} + \cdots + \frac{1}{2^{r_s}} \right) \right] \\ &\quad + 2^{2r-p-3}(2^p - 1)(2^{r_{s+1}} + 2^{r_{s+2}} + \cdots + 2^{r_n}). \end{aligned}$$

Proof. Label the vertices of G using Lemma 2.1 from 1 to 2^r. Since the graph H contains an n-dimensional grid and label the vertices of n-dimensional grid using Lemma 2.2 from 1 to 2^r. For illustration, see Figures 2, 3, 4, 5 and 6. Let us assume that, the label represent each of the vertex, which is allocated by the above algorithms. Let $f : G \to H$ be an embedding and let $f(v) = v$

for all $v \in V(G)$ and for $uv \in E(G)$, let $P_f(uv)$ be a path (shortest) between $f(u)$ and $f(v)$ in H. Now, we have the following 3 cases.

Case 1. $s = n$

It is clear that, the graph H becomes an n-dimensional grid $P_{2^{r_1}} \square P_{2^{r_2}} \square \cdots \square P_{2^{r_n}}$. For all i, j, $1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i} - 1$, let S_i^j be the edge cut of $P_{2^{r_1}} \square P_{2^{r_2}} \square \cdots \square P_{2^{r_n}}$ consisting of the edges between the j^{th} and $(j+1)^{th}$ copies of $P_{2^{r_1}} \square P_{2^{r_2}} \square \cdots \square P_{2^{r(i-1)}} \square P_{2^{r(i+1)}} \square \cdots \square P_{2^{r_n}}$. Then $\{S_i^j : 1 \leq i \leq n \text{ and } 1 \leq j \leq 2^{r_i} - 1\}$ is an edge-partition of $P_{2^{r_1}} \square P_{2^{r_2}} \square \cdots \square P_{2^{r_n}}$.

For all i, j, $1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i} - 1$, $E(H) \backslash S_i^j$ has two components H_i^j and $\overline{H}_i^j$, where $|V(H_i^j)| = (2^{r-r_i})j$ and $|V(\overline{H}_i^j)| = 2^{r-r_i}(2^{r_i} - j)$. Let G_i^j and $\overline{G}_i^j$ be the induced subgraph of the inverse images of $V(H_i^j)$ and $V(\overline{H}_i^j)$ under f respectively. By Lemma 2.1, $deg_G(v) = 2^{r-r_i-p}(2^p - 1)j$, for all $v \in V(G_i^j)$ and hence $I_G((2^{r-r_i})j) = 2^{2r-2r_i-p}(2^p - 1)j^2/2$. By Case 2 of Lemma 1.3, $E(G_i^j)$ is the maximum subgraph on $|V(G_i^j)| = (2^{r-r_i})j$ vertices in G. Thus the edge cut S_i^j fulfil all the conditions of Lemma 1.1. Therefore

$$\begin{aligned} EC_f(S_i^j) &= 2^{r-p}(2^p - 1)(2^{r-r_i})j - 2\left(\frac{2^{2r-2r_i-p}(2^p - 1)j^2}{2}\right) \\ &= 2^{2r-2r_i-p}(2^p - 1)(2^{r_i} - j)j \end{aligned}$$

is minimum for $1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i} - 1$.
Then by Lemma 1.2,

$$\begin{aligned} WL(G, H) &= \sum_{i=1}^{n} \sum_{j=1}^{2^{r_i}-1} EC_f(S_i^j) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{2^{r_i}-1} 2^{2r-2r_i-p}(2^p - 1)(2^{r_i} - j)j \\ &= \frac{2^{2r-p}(2^p - 1)}{6}\left[(2^{r_1} + 2^{r_2} + \cdots + 2^{r_n}) - \left(\frac{1}{2^{r_1}} + \frac{1}{2^{r_2}} + \cdots + \frac{1}{2^{r_n}}\right)\right]. \end{aligned}$$

Case 2. $s = 0$

It is clear that, the graph H becomes an n-dimensional torus $C_{2^{r_1}} \square C_{2^{r_2}} \square \cdots \square C_{2^{r_n}}$. For all i, j, $1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i-1}$, let T_i^j be the edge cut of $C_{2^{r_1}} \square C_{2^{r_2}} \square \cdots \square C_{2^{r_n}}$ consisting of the edges between the $(2^{r_i-1} - i + j)^{th}$ & $(2^{r_i-1} - i + j + 1)^{th}$ and $(2^{r_i} - i + j)^{th}$ & $(2^{r_i} - i + j + 1)^{th}$ copies of $C_{2^{r_1}} \square C_{2^{r_2}} \square \cdots \square C_{2^{r(i-1)}} \square P_{2^{r_i-1}} \square C_{2^{r(i+1)}} \square \cdots \square C_{2^{r_n}}$. Then $\{T_i^j : 1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i-1}\}$ is an edge-partition of $C_{2^{r_1}} \square C_{2^{r_2}} \square \cdots \square C_{2^{r_n}}$.

For all i, j, $1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i-1}$, $E(H) \backslash T_i^j$ has two components H_i^j and $\overline{H}_i^j$, where $|V(H_i^j)| = |V(\overline{H}_i^j)| = 2^{r-1}$. Let G_i^j and $\overline{G}_i^j$ be the induced subgraph of the inverse images of $V(H_i^j)$ and $V(\overline{H}_i^j)$ under f respectively. By Lemma 2.1, $deg_G(v) = 2^{r-p-1}(2^p - 1)$, for all $v \in V(G_i^j)$ and hence $I_G(2^{r-1}) = 2^{2r-2p-2}2^p(2^p - 1)/2$. By Case 2 of Lemma 1.3, $E(G_i^j)$ is the maximum subgraph on $|V(G_i^j)| = 2^{r-1}$ vertices in G. Thus the edge cut T_i^j fulfil all the conditions of Lemma 1.1. Therefore

$$\begin{aligned} EC_f(T_i^j) &= 2^{r-p}(2^p-1)2^{r-1} - 2\left(\frac{2^{2r-2p-2}2^p(2^p-1)}{2}\right) \\ &= 2^{2r-p-2}(2^p-1) \end{aligned}$$

is minimum for $1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i-1}$.
Then by Lemma 1.2,

$$\begin{aligned} WL(G,H) &= \sum_{i=1}^{n}\sum_{j=1}^{2^{r_i-1}} EC_f(T_i^j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{2^{r_i-1}} 2^{2r-p-2}(2^p-1) \\ &= 2^{2r-p-3}(2^p-1)(2^{r_1}+2^{r_1}+\cdots+2^{r_n}). \end{aligned}$$

Case 3. $n > s > 0$
In this case, we describe H as the Cartesian product of $n \geq 3$ factors of respective order 2^{r_i}, $i \in [n]$ where $i \in [n]$, where $r_1 \leq r_2 \leq \cdots \leq r_n$, $r_1 + r_2 + \cdots + r_n = r$ and each factor is a path or a cycle, $r \geq 3, 1 \leq p < r$.
Let $s > 0$ factors of H are paths and the other $(n-s)$ factors are cycles, then obtained by using the associativity of the Cartesian product and writing that,

$$P_{2^{r_1}}\square\cdots P_{2^{r_s}}\square C_{2^{r_{s+1}}}\square\cdots\square C_{2^{r_n}} = (P_{2^{r_1}}\square\cdots\square P_{2^{r_s}})\square(C_{2^{r_{s+1}}}\square\cdots\square C_{2^{r_n}})$$

Let S_i^l, $1 \leq i \leq s$, $1 \leq l \leq 2^{r_i}-1$ and T_i^m, $s+1 \leq i \leq n$, $1 \leq m \leq 2^{r_i-1}$ be the edge cuts of paths and cycles of H respectively.

By similar arguments in Case 1 and Case 2, we get

$$EC_f(S_i^l) = 2^{2r-2r_i-p}(2^p-1)(2^{r_i}-l)l$$

is minimum for $1 \leq i \leq s$ and $1 \leq l \leq 2^{r_i}-1$ and

$$EC_f(T_i^m) = 2^{2r-p-2}(2^p-1)$$

is minimum for $s+1 \leq i \leq n$ and $1 \leq m \leq 2^{r_i-1}$.

Then by Lemma 1.2,

$$\begin{aligned} WL(G,H) &= \sum_{i=1}^{s}\sum_{l=1}^{2^{r_i}-1} EC_f(S_i^l) + \sum_{i=s+1}^{n}\sum_{m=1}^{2^{r_i-1}} EC_f(T_i^m) \\ &= \sum_{i=1}^{s}\sum_{l=1}^{2^{r_i}-1} 2^{2r-2r_i-p}(2^p-1)(2^{r_i}-l)l + \sum_{i=s+1}^{n}\sum_{m=1}^{2^{r_i-1}} 2^{2r-p-2}(2^p-1)(2^{2r_i}-1) \\ &= \frac{1}{6}\sum_{i=1}^{s} 2^{2r-r_i-p}(2^p-1)(2^{r_i}-1)(2^{r_i}+1) + \sum_{i=s+1}^{n} 2^{2r+r_i-p-3}(2^p-1) \\ &= \frac{2^{2r-p}(2^p-1)}{6}\left[(2^{r_1}+2^{r_2}+\cdots+2^{r_s}) - \left(\frac{1}{2^{r_1}}+\frac{1}{2^{r_2}}+\cdots+\frac{1}{2^{r_s}}\right)\right] \\ &\quad +2^{2r-p-3}(2^p-1)(2^{r_{s+1}}+2^{r_{s+2}}+\cdots+2^{r_n}). \end{aligned}$$

□

Corollary 2.1. *If G_1 is a path on 2^{r_1} vertices and G_i is a cycle on 2^{r_i} vertices, $2 \le i \le n$, then the host graph becomes an n-dimensional cylinder $P_{2^{r_1}} \square C_{2^{r_2}} \square \cdots \square C_{2^{r_n}}$ and the wirelength of an embedding f from G into H is given by*

$$WL(G,H) = \frac{2^{2r-p}(2^p-1)}{6}\left[2^{r_1} - \frac{1}{2^{r_1}}\right] + 2^{2r-p-3}(2^p-1)(2^{r_2}+2^{r_3}+\cdots+2^{r_n}).$$

3. Conclusion and Future Work

In this manuscript, we found the wirelength (exact and minimum) of an embedding complete multi-partite graphs into Cartesian product of paths and/or cycles. Computing the dilation [4] and the edge congestion of embedding complete multi-partite graphs into Cartesian product and other product of graphs are under investigation.

Acknowledgement

The work of R. Sundara Rajan was partially supported by Project no. ECR/2016/1993, Science and Engineering Research Board (SERB), Department of Science and Technology (DST), Government of India. Further, we thank Prof. Gregory Gutin and Prof. Stefanie Gerke, Royal Holloway, University of London, TW20 0EX, UK; Prof. Indra Rajasingh, School of Advanced Sciences, Vellore Institute of Technology, Chennai, India and Dr. N. Parthiban, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India, for their fruitful suggestions. Further, the authors would like to thank the anonymous referees for their comments and suggestions. These comments and suggestions were very helpful for improving the quality of this paper.

References

[1] D. Adolphson and T.C. Hu, Optimal linear ordering, *SIAM J. Appl Math.* **25**, 403–423, 1973.

[2] M. Arockiaraj, J.N. Delaila, and J. Abraham, Optimal wirelength of balanced complete multipartite graphs onto cartesian product of {Path, Cycle} and trees, Fundam. Inform. **178** (2021), 187-202.

[3] L. Auletta, A.A. Rescigno, and V. Scarano, Embedding graphs onto the supercube, *IEEE Trans Comput*. **44** (1995), 593–597.

[4] S.L. Bezrukov, J.D. Chavez, L.H. Harper, M. Röttger, and U.P. Schroeder, Embedding of hypercubes into grids, *MFCS. Lect. Notes Comput. Sci.* **1450** (1998), 693–701.

[5] S.L. Bezrukov, S.K. Das, and R. Elsässer, An edge-isoperimetric problem for powers of the Petersen graph, *Ann. Comb.* **4** (2000), 153–169.

[6] S.L. Bezrukov and U.P. Schroeder, The cyclic wirelength of trees, *Discrete. Appl. Math.* **87** (1998), 275–277.

[7] M.Y. Chan, Embedding of grids into optimal hypercubes, *SIAM J. Comput.* **20** (1991), 834–864.

[8] M.Y. Chan and S.J. Lee, Fault-tolerant embedding of complete binary trees in hypercubes, *IEEE Trans Parallel Distrib Syst*. **4** (1993), 277–288.

[9] B. Cheng, J. Fan, and X. Jia, Dimensional-Permutation-Based independent spanning trees in bijective connection networks, *IEEE Trans Parallel Distrib Syst*. **26** (2015), 45-53.

[10] B. Cheng, D. Wang, and J. Fan, Constructing completely independent spanning trees in crossed cubes, *Discrete Appl. Math.* **219** (2017), 100-109.

[11] F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg, Embedding graphs in books: A layout problem with applications to VLSI design, *SIAM J. Alg. Discrete Math.* **8** (1987), 33-58.

[12] S.A. Choudum and U. Nahdini, Complete binary trees in folded and enhanced cubes, *Networks*, **43** (2004), 266–272.

[13] J. Diaz, J. Petit, and M. Serna, A Survey of Graph Layout Problems, *ACM Comput. Surv.* **34** (2002), 313–356.

[14] J. Ellis, S. Chow, and D. Manke, Many to one embeddings from grids into cylinders, tori and hypercubes, *SIAM J. Comput.* **32** (2003), 386–407.

[15] J. Fan, X. Lin, X. Jia, and R.W.H. Lau, *Edge-Pancyclicity of twisted Cubes*, *(ISAAC). Lect. Notes Comput. Sci (LNCS).* **3827** (2005), 1090-1099.

[16] J.-F. Fang and K.-C. Lai, Embedding the incomplete hypercube in books, *Inf. Process. Lett.* **96** (2005), 1–6.

[17] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, (1979).

[18] C.-J. Guu, *The Circular Wirelength Problem for Hypercubes*, Ph.D. dissertation, University of California, Riverside, (1997).

[19] L.H. Harper, *Global Methods for Combinatorial Isoperimetric Problems*, Cambridge University Press, (2004).

[20] M.H. Khalifeh, H.Y.-Azari, and A.R. Ashrafi, The hyper-Wiener index of graph operations, *Comput. Math. with Appl.* **56** (2008), 1402–1407.

[21] P. Kulasinghe, and S. Bettayeb, Embedding binary trees into crossed cubes, *IEEE Trans Comput.* **44** (1995), 923–929.

[22] K.J. Kumar, S. Klavžar, R.S. Rajan, I. Rajasingh, and T.M. Rajalaxmi, An asymptotic relation between the wirelength of an embedding and the wiener index, Discrete Math. Lett., to appear.

[23] P.-L. Lai and C.-H. Tsai, Embedding of tori and grids into twisted cubes, *Theor. Comput. Sci.* **411** (2010), 3763–3773.

[24] R.-S. Lo and G.-H. Chen, Embedding Hamiltonian paths in faulty arrangement graphs with the backtracking method, *IEEE Trans Parallel Distrib Syst*. **12** (2001), 209–222.

[25] P. Manuel, I. Rajasingh, B. Rajan, and H. Mercy, Exact wirelength of hypercube on a grid, *Discret. Appl. Math.* **157** (2009), 1486–1495.

[26] R.S. Mary and I. Rajasingh, Embedding of complete graphs and cycles into grids with holes, *Procedia Comput. Sci.* **172** (2020), 115–121.

[27] M. Miller, R.S. Rajan, N. Parthiban, and I. Rajasingh, Minimum linear arrangement of incomplete hypercubes, *J. Comput.* **58** (2015), 331–337.

[28] G.K. Nandini, R.S. Rajan, T.M. Rajalaxmi, A.A. Shantrinal, K.S.K. Sharifah, and H. Roslan, Wiener index via wirelength of an embedding, *Discrete Math. Algorithms Appl.* (2021).

[29] R.S. Rajan, T. Kalinowski, S. Klavžar, H. Mokhtar, and T.M. Rajalaxmi, Lower bounds for dilation, Wirelength, and edge Congestion of embedding graphs into hypercubes, J Supercomput. **77** (2021), 4135-4150.

[30] R.S. Rajan, T.M. Rajalaxmi, J.B. Liu, and G. Sethuraman, Wirelength of embedding complete multipartite graphs into certain graphs, *Discrete Appl. Math.* **280** (2020), 221-236.

[31] R.S. Rajan, T.M. Rajalaxmi, S. Stephen, A.A. Shantrinal, and K. Jagadeesh Kumar, Embedding wheel-like networks, *Iran. J. Math. Sci. Inform.* (2020).

[32] I. Rajasingh, M. Arockiaraj, B. Rajan, and P. Manuel, Minimum wirelength of hypercubes into n-dimensional grid networks, *Inf. Process. Lett.* **112** (2012), 583–586.

[33] I. Rajasingh, J. Quadras, P. Manuel, and A. William, Embedding of cycles and wheels into arbitrary trees, *Networks*, **44** (2004), 173–178.

[34] I. Rajasingh and R.S. Rajan, Exact wirelength of embedding circulant networks into necklace and windmill Graphs, *Ars Comb.* **130** (2017), 215–237.

[35] I. Rajasingh, R.S. Rajan, and P. Manuel, A linear time algorithm for embedding Christmas trees into certain trees, *Parallel Process. Lett.* **25** (2015), 1–17.

[36] I. Rajasingh, B. Rajan, and R.S. Rajan, Embedding of special classes of circulant networks, hypercubes and generalized Petersen graphs, *Int. J. Comput. Math.* **89** (2012), 1970–1978.

[37] H. Rostami and J. Habibi, Minimum linear arrangement of chord graphs, *Appl. Math. Comput.* **203** (2008), 358–367.

[38] B.H. Susanti, A.N.M. Salman, and R. Simanjuntak, The rainbow 2-connectivity of Cartesian products of 2-connected graphs and paths, *Electron. J. Graph Theory Appl. 8 (2020), 145–156.*

[39] Y.-C. Tseng, S.-H. Chang, and J.-P. Sheu, Fault-tolerant ring embedding in a star graph with both link and node failures, *IEEE Trans Parallel Distrib Syst*. **8** (1997), 1185–1195.

[40] A. Wagner and D.G. Corneil, Embedding trees in a hypercube is np-complete, *SIAM J. Comput.* **19** (1990), 570–590.

[41] X. Wang, A. Erickson, J. Fan, and X. Jia, Hamiltonian properties of DCell networks, *J. Comput.* **58** (2015), 2944-2955.

[42] M.-C. Yang, Path embedding in star graphs, *Appl. Math. Comput.* **207** (2009), 283–291.

[43] X. Yang, Q. Dong, and Y.Y. Tan, Embedding meshes/tori in faulty crossed cubes, *Inf. Process. Lett.* **110** (2010), 559–564.

[44] P.-J. Yang, S.-B. Tien, and C.S. Raghavendra, Embedding of rings and meshes onto faulty hypercubes using free dimensions, *IEEE Trans. Comput.* **43** (1994), 608–613.

[45] J.M. Xu, *Topological Structure and Analysis of Interconnection Networks*, Kluwer Academic Publishers, (2001).

Annexure I

Python program for labeling of the guest graph

```
def printline(num, boxes, boxesinrow):
    i = 0;
    string = ' '
    for i in range(0, boxesinrow):
      for j in range(0, 4):
        string = string + "{:< 3d} ".format(num + boxes * j) + ' '
      string = string + ' '
      num = num + 1
      print(string)
def printbox(num, boxes, elements_ per_box, boxes_ in_row):
    value=num
    temp = elements_ per_ box // 4
    while temp > 0:
      printline(num, boxes, boxes_ in_ row)
      temp = temp - 1
      num = num + boxes * 4
    print('\n')
    return (value+boxes_ in_ row)
def printpattern(numl):
    boxes = len(numl)
    elements_ Per_ box = numl[0]
    num = 1
    temp = boxes
    while temp > 0:
      if temp >= 4:
        num = printbox(num, boxes, elements_ Per_ box, 4)
      else:
        num = printbox(num, boxes, elements_ Per_ box, temp)
      temp = temp - 4
printpattern([32,32,...,32])
```

Implementation of the above Python program

Output 1:

I

1	9	17	25
33	41	49	57
65	73	81	89
97	105	113	121
129	137	145	153
161	169	177	185
193	201	209	217
225	233	241	249

II

2	10	18	26
34	42	50	58
66	74	82	90
98	106	114	122
130	138	146	154
162	170	178	186
194	202	210	218
226	234	242	250

III

3	11	19	27
35	43	51	59
67	75	83	91
99	107	115	123
131	139	147	155
163	171	179	187
195	203	211	219
227	235	243	251

IV

4	12	20	28
36	44	52	60
68	76	84	92
100	108	116	124
132	140	148	156
164	172	180	188
196	204	212	220
228	236	244	252

V

5	13	21	29
37	45	53	61
69	77	85	93
101	109	117	125
133	141	149	157
165	173	181	189
197	205	213	221
229	237	245	253

VI

6	14	22	30
38	46	54	62
70	78	86	94
102	110	118	126
134	142	150	158
166	174	182	190
198	206	214	222
230	238	246	254

VII

7	15	23	31
39	47	55	63
71	79	87	95
103	111	119	127
135	143	151	159
167	175	183	191
199	207	215	223
231	239	247	255

VIII

8	16	24	32
40	48	56	64
72	80	88	96
104	112	120	128
136	144	152	160
168	176	184	192
200	208	216	224
232	240	248	256

Figure 3: Complete 8-partite graph $K_{32,32,\ldots,32}$

Output 2:

Set				
I	1	33	65	97
	129	161	193	225
II	2	34	66	98
	130	162	194	226
III	3	35	67	99
	131	163	195	227
IV	4	36	68	100
	132	164	196	228
V	5	37	69	101
	133	197	101	229
VI	6	38	70	102
	134	166	198	230
VII	7	39	71	103
	135	167	199	231
VIII	8	40	72	104
	136	168	200	232
IX	9	41	73	105
	137	169	201	233
X	10	42	74	106
	138	170	202	234
XI	11	43	75	107
	139	171	203	235
XII	12	44	76	108
	140	172	204	236
XIII	13	45	77	109
	141	173	205	237
XIV	14	46	78	110
	142	174	206	238
XV	15	47	79	111
	143	175	207	239
XVI	16	48	80	112
	144	176	208	240
XVII	17	49	81	113
	145	177	209	241
XVIIII	18	50	82	114
	146	178	210	242
XIX	19	51	83	115
	147	179	211	243
XX	20	52	84	116
	148	180	212	244
XXI	21	53	85	117
	149	181	213	245
XXII	22	54	86	118
	150	182	214	246
XXIII	23	55	87	119
	151	183	215	247
XXIV	24	56	88	120
	152	184	216	248
XXV	25	57	89	121
	153	185	217	249
XXVI	26	58	90	122
	154	186	218	250
XXVII	27	59	91	123
	155	187	219	251
XXVIII	28	60	92	124
	156	188	220	252
XXIX	29	61	93	125
	157	189	221	253
XXXX	30	62	94	126
	158	190	222	254
XXXI	31	63	95	127
	159	191	223	255
XXXII	32	64	96	128
	160	192	224	256

Figure 4: Complete 32-partite graph $K_{8,8,\ldots,8}$

Annexure II

Python program for labeling of the host graph

```
n=0
def disp_3nr(numl, n, irange):
    for i in irange:
        string = str(i+n)
        for j in range(1, numl[1]):
            string=string+' '+str(i+n+numl[0]*j)
        for k in range(1, numl[2]):
            string=string+' '
        for j in range(0, numl[1]):
                string = string + ' ' + str(i+ n + numl[0] * j+ k*numl[0]*numl[1])
        print(string)
def disp_ n(numl):
    base=numl[0]*numl[1]*numl[2]
    para_ num=len(numl)
    order=numl[3:]
    global n
    n = -base
    tnum1=numl[:3]
    loopri(order, tnum1, base)
def rotate(irange):
    temp=irange[0]
    for i in range(0, len(irange)-1):
        irange[i] = irange[i+1]
    irange[len(irange)-1]=temp
    return(irange)
def loopri(order, tnuml, base):
    irange = list(range(1, tnuml[0]+1))
    if len(order) == 1:
        for i in range(0, order[0]):
            global n
            print(i+1)
            n = n + base
            disp_ 3nr(tnuml, n, irange)
            irange=rotate(irange)
            print(' ')
    else:
        for i in range(0, order[-1]):
            print(i)
            loopr(order[:-1], tnuml, base, irange)
            irange=rotate(irange)
def loopr(order, tnuml, base, irange):
    if len(order) == 1:
        for i in range(0, order[0]):
            global n
            n = n + base
            disp_ 3nr(tnuml, n, irange)
            print(' ')
    else:
        for i in range(0, order[-1]):
            print(i)
            loopr(order[:-1], tnuml, base, irange)
disp_ n([4,8,16])
```

Implementation of the above Python program

Output 1:

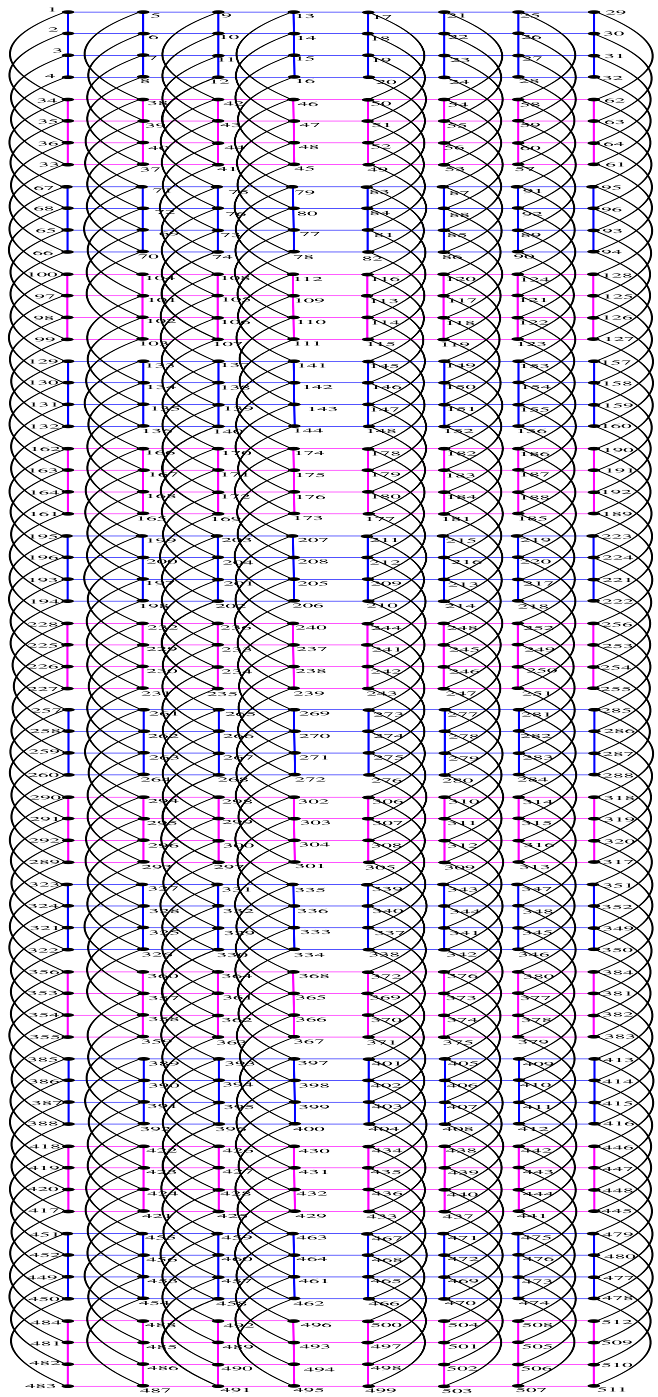

Figure 5: 3-dimensional grid $P_4 \square P_8 \square P_{16}$

Output 2:

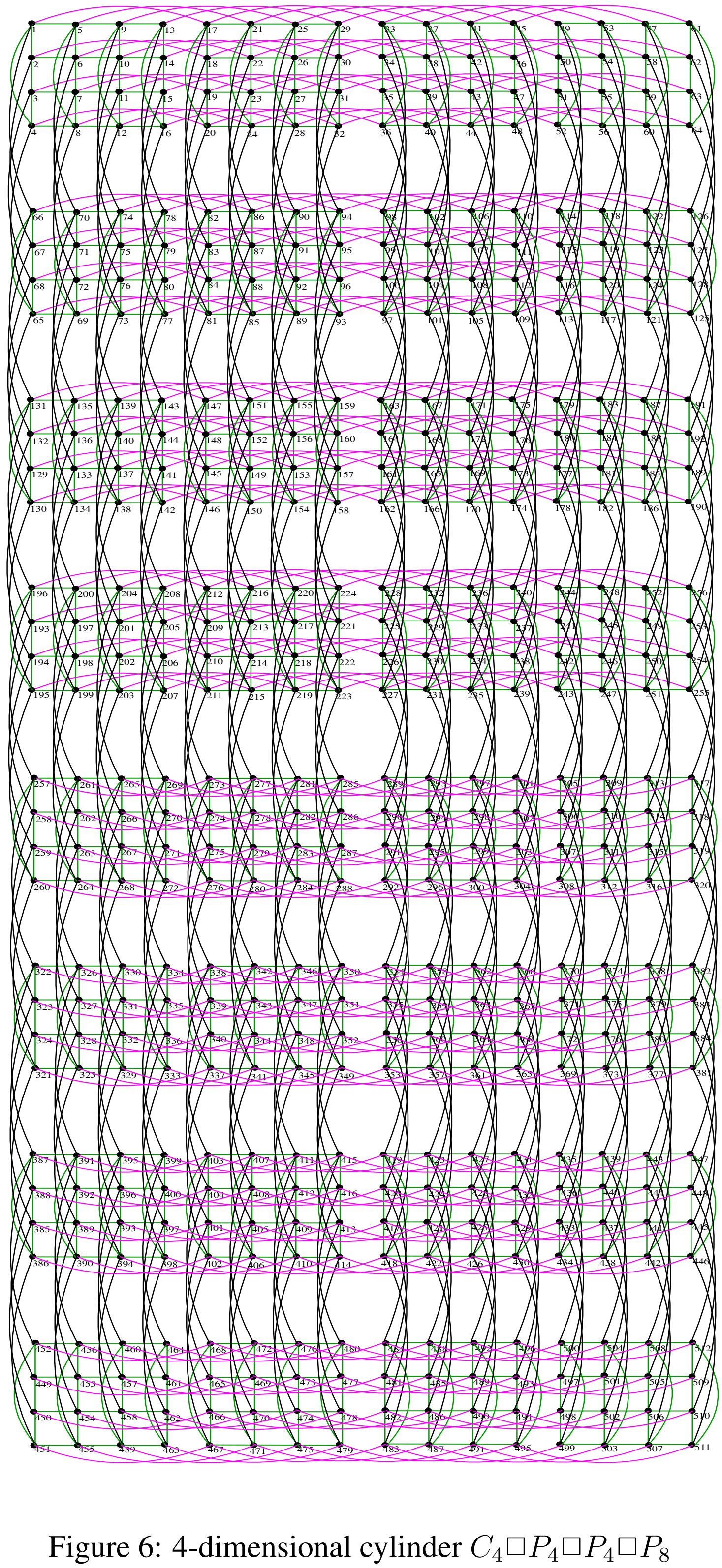

Figure 6: 4-dimensional cylinder $C_4 \square P_4 \square P_4 \square P_8$