Optimizing Traffic Locality using Hilbert Curve Mapping

Nikolaos Lazaridis Polyvios Pratikakis Fabien Chaix

Department of Computer Science, University of Crete

Abstract

In high-performance computing systems, communication efficiency is critically dependent on traffic locality (placing frequently communicating nodes physically close). Poor locality leads to increased latency and energy consumption. This report investigates the effectiveness of Hilbert space-filling curve mapping to improve traffic locality on 2D and 3D networks. We developed a C++ analysis tool to model torus architecture of arbitrary dimensions , and evaluate mapping quality. This evaluation compares communication cost before and after Hilbert reordering by calculating traffic-weighted distances using Euclidean, Manhattan, and Chebyshev distance metrics. This analysis demonstrates the utility of Hilbert curves for improving communication locality, although its effectiveness varies depending on the specific communication pattern being mapped.

1 Introduction

High-performance computing (HPC) is a field that has the mission of grouping computing resources to solve problems that are beyond the complexity or computational demands of one regular computer. An HPC system can be a custom supercomputer or, more commonly, an aggregation of many interconnected individual computers, referred to as nodes. These nodes are typically specialized like controller nodes for coordination, login/interactive nodes for user access and a very large number of compute or worker nodes that are actually performing the parallel computations. Parallel software and algorithms are assigned on these nodes, each contributing an input to a bigger task.

As mentioned, the network hardware is a critical component of HPC systems that enable communication between compute nodes. One popular and well-studied architectural choice for this interconnect is the **Torus topology** [1]. Conceptually, a torus network arranges nodes in a multi-dimensional grid, most often two-dimensional (2D) or three-dimensional (3D). The defining characteristic of the torus is the existence of **wrap-around connections**. These edges link the nodes along the periphery of the grid to nodes on the opposite periphery. For instance, in a size X $\times$ Y $\times$ Z, 3D torus grid, the node with coordinates (X-1, y, z) is linked directly to node (0, y, z), and comparable links exist in the Y and Z directions as well. This actually creates a network with no boundaries or physical edges, similar to the surface of a donut (for 2D) or its higher-dimensional counterparts but it can also be depicted as a mesh.

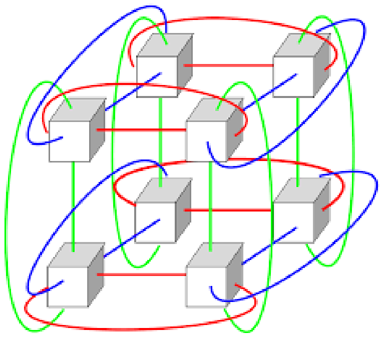


Figure 1: Diagram illustrating a 3D Mesh Torus Network

Nevertheless, optimal performance on a torus is heavily dependent on efficiently mapping communicating tasks to reduce communication distances. One way to generate locality-sensitive mappings is by using mathematical constructs known as space-filling curves.

The theory of space-filling curves was developed in the late 19th century. In 1890, Giuseppe Peano proved the counter-intuitive fact that there is a continuous curve passing through all points of a unit square and thus mapping a 1D interval onto a 2D area [2]. Following this, David Hilbert later defined another construction for such a curve in 1891, now the Hilbert curve [3].

The Hilbert curve is a continuous, fractal curve whose path fills a higher-dimensional space, like a unit square or cube. Its construction is typically defined recursively, generating a sequence of piecewise linear curves that more and more closely approach the final space-filling limit. Visualizing the early iterations of these curves helps illustrate their complex nature.

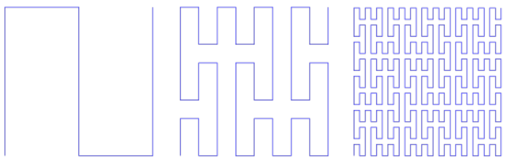


Figure 2: First few iterations of the Peano curve construction.

What makes the Hilbert curve especially well-suited for so many various computational uses is its improved **locality-preserving behavior** compared to many other space-filling curves, including the original Peano curve. Neighboring points in the N-dimensional space will tend to be close to each other in the 1-dimensional sequence generated by the Hilbert curve walk [4]. This property is particularly valuable for functions such as multi-dimensional data indexing and, as researched in this report, constructing potentially locality-sensitive mappings of nodes in a network.

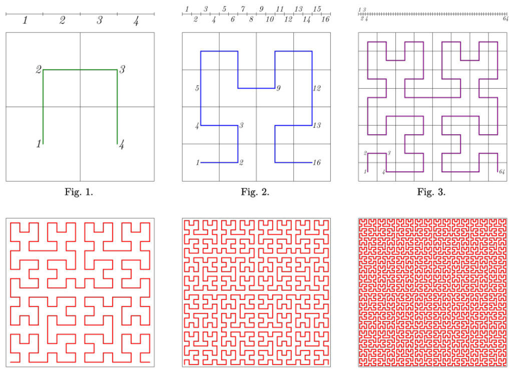


Figure 3: First few iterations of the Hilbert curve construction.

Like most space-filling curves, the Hilbert curve is usually defined in an iterative way, as the limit of a sequence of piecewise linear approximations. The algorithm employed to create the curve and transform coordinates into Hilbert indices is detailed further in Section 3.1.

To illustrate the structure of communication in different parallel workloads, Figures 4 and 5 show the raw communication patterns for two representative cases: "CG" and "SP". Each pixel in these 256×256 heatmaps represents a communication link between a pair of nodes, with brighter colors (e.g., yellow) indicating stronger or more frequent communication. In Figure 4, the "CG" pattern exhibits a diagonal-heavy structure, suggesting that many node pairs communicate with others that are nearby in index space—hinting at spatial locality. By contrast, Figure 5 reveals that the "SP" pattern has a more dispersed and irregular communication profile, with connections spanning longer distances in the index space. These visualizations are key for understanding why certain mappings, like the Hilbert curve, might improve or degrade performance based on how well they preserve or disrupt these intrinsic structures.

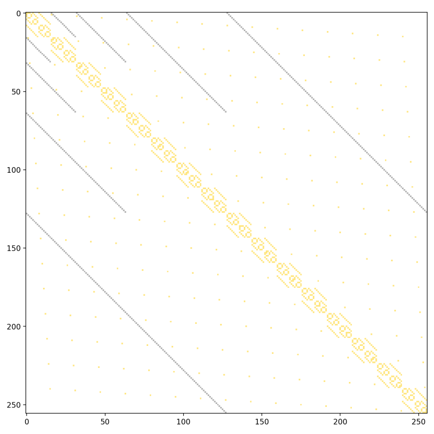


Figure 4: Communication pattern for the "CG" workload. The diagonal structure suggests strong locality in node interactions.

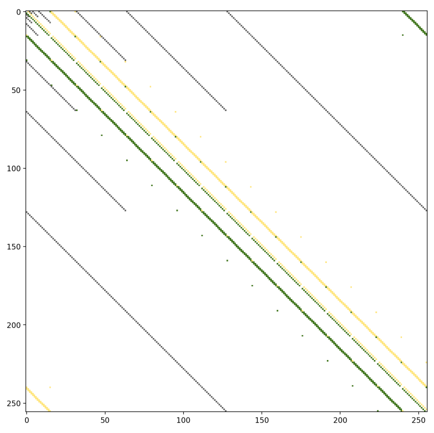


Figure 5: Communication pattern for the "SP" workload. The dispersed structure indicates longer-range, irregular node interactions.

2 Background and Related Work

2.1 How Hilbert Curve is Constructed

The locality-preserving and space-filling property of Hilbert curve is due to its recursive structure. We can learn about the principle behind it from its iterative construction process:

1. **Base Case (Order 1):** The construction starts with a simple base pattern. For example, in two dimensions (2D), the order 1 Hilbert curve can be visualized as a 'U' shape connecting the centers of the four quadrants of a unit square in a specific sequence (e.g., bottom-left to top-left to top-right to bottom-right).

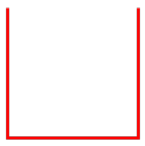


Figure 6: First Order of Hilbert Curve before rotation.

2. **Recursive Step (Order N $\rightarrow$ Order N+1):** To form the next higher-order curve (order N+1), the present space (say the unit square) is divided into smaller regions of a similar shape (usually 2^d regions in dimension d, say four quadrants in 2D, eight octants in 3D). Each of these regions will have a smaller-sized copy of the order N Hilbert curve.

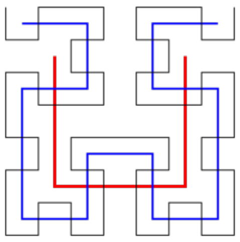


Figure 7: First to Third Order of Hilbert Curve before rotation.

3. **Orientation and Connection:** This is the step of continuity and locality. The smaller order N curves that we put into the subdivisions aren't identical copies; rather, they're **oriented and/or reflected** according to pre-specified rules. These rules are designed such that the endpoint of the curve in one subdivision connects precisely to the starting point of the curve in the logically "next" subdivision, according to the overall path of the order N+1 curve. This ensures the resulting higher-order curve is continuous and avoids large jumps across the space.

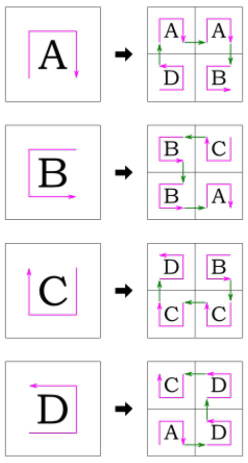


Figure 8: Hilbert Curve Rules.

4. **Limit:** The true Hilbert curve is the mathematical limit approached as this recursive process is repeated infinitely. At each iteration, the curve becomes longer and more intricate, eventually passing arbitrarily close to every point within the initial space (e.g., the unit square or cube).

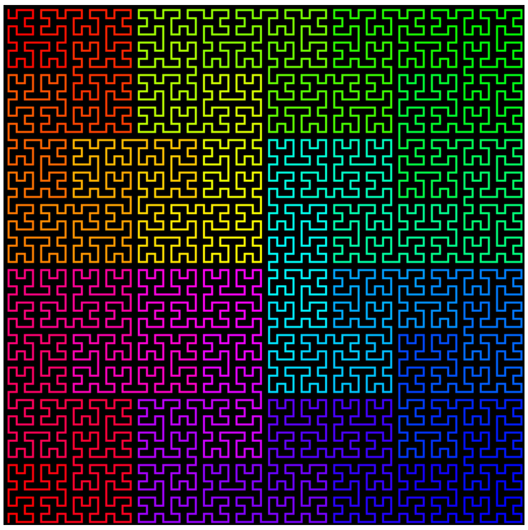


Figure 9: Hilbert Curve with the colour showing the Locality property.

2.2 Mathematical and Topological View of the Torus

While we often visualize a torus network as a grid (Figure 1), mathematically and topologically, it's useful to understand its structure in two primary ways:

1. **The Torus as a Quotient Space (Grid with Wrap-Around):** From a network connectivity perspective, a d-dimensional torus with side lengths $L_1, L_2, ..., L_d$ can be formally defined as a quotient space of a d-dimensional Euclidean grid. Consider a grid of points $(x_1, x_2, ..., x_d)$ where each x_i is an integer $0 \leq x_i < L_i$. The torus topology is created by identifying opposite edges of this grid. This identification, or "wrap-around," means that a point $(x_1, ..., x_{i-1}, L_i - 1, x_{i+1}, ..., x_d)$ is considered adjacent (connected) to the point $(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_d)$ for each dimension i.

   Mathematically, this is equivalent to defining coordinates using modular arithmetic. Two points $(x_1, ..., x_d)$ and $(y_1, ..., y_d)$ represent the same node in the torus if $x_i \equiv y_i \pmod{L_i}$ for all $i = 1, ..., d$. The connections are then between nodes whose coordinates differ by exactly ± 1 in one dimension (modulo L_i). This perspective emphasizes the grid structure and the wrap-around links, which is most relevant for calculating network distances like hop count (Manhattan distance) and understanding the interconnect wiring, as depicted in Figure 1.

2. **The Torus as a Topological Surface (Doughnut Shape):** Geometrically, the term "torus" often refers to the specific doughnut-shaped surface embedded in three-dimensional space ($\mathbb{R}^3$). This surface can be generated by revolving a circle around an axis in its plane, where the axis does not intersect the circle.

   A standard parametrization uses two angles, θ and ϕ (both typically ranging from 0 to 2π), representing rotation around the tube and rotation around the torus's main axis of revolution, respectively. Let R be the **major radius** (distance from the center of the torus hole to the center of the tube) and r be the **minor radius** (the radius of the tube itself). The coordinates (x, y, z) of a point on the torus surface are then given by **placeholder˙torus˙param**:

$$
\begin{aligned}
x(\theta, \phi) &= (R + r\cos\theta)\cos\phi \\
y(\theta, \phi) &= (R + r\cos\theta)\sin\phi \\
z(\theta, \phi) &= r\sin\theta
\end{aligned}
$$

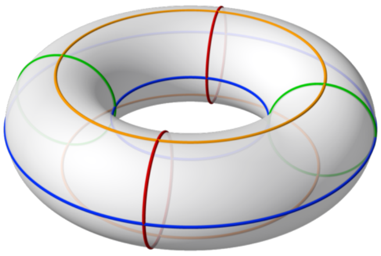


Figure 10: Diagram illustrating a 3D Torus Doughnut Network

2.3 Defining Traffic Locality and Communication Cost

Traffic Locality: Traffic locality refers to the spatial distribution of communication patterns relative to the physical layout of the network.

- **Good Locality:** Occurs when nodes that frequently exchange data are mapped to physical locations that are topologically "close" within the network.
- **Poor Locality:** Occurs when nodes with high communication volume are mapped to physical locations that are topologically "far apart".

A network mapping is considered *locality-aware* if it attempts to position nodes in a way nodes such that common communication patterns result in short paths through the network. Mathematically, for a single pair of communicating nodes, i.e., node u and node v, the minimum communication cost is typically modeled as a function of the physical distance between their locations in the network:

$$\text{Cost}(u, v) = \text{Distance}(P_u, P_v)$$

where P_u and P_v are the physical coordinates (e.g., (x_u, y_u, z_u) and (x_v, y_v, z_v)) of nodes u and v in the torus grid, and $\text{Distance}(\cdot, \cdot)$ is a metric appropriate for the torus topology (like the Manhattan, Euclidean, or Chebyshev distances). For a torus, these distance metrics must account for the wrap-around links:

$$\begin{aligned}\text{Distance}((x_u, y_u, z_u), (x_v, y_v, z_v)) = f(\, &\min(|x_u - x_v|, X - |x_u - x_v|),\\ &\min(|y_u - y_v|, Y - |y_u - y_v|),\\ &\min(|z_u - z_v|, Z - |z_u - z_v|))\end{aligned}$$

where X, Y, Z are the dimensions of the torus, and f is a function specific to the metric (sum for Manhattan, square root of sum of squares for Euclidean, max for Chebyshev).

To calculate the network-wide communication efficiency for an application with a pre-specified **traffic pattern**, we need to take into account how much **data** is communicated between nodes in a pair. Traffic between high-traffic volume pairs of far-apart nodes adds much more to the overall network load compared to their near-node or low-volume counterparts.

Therefore, a more relevant metric for overall communication cost is the **traffic-weighted distance**. For a specific pair of nodes (u, v) with communication volume $T(u, v)$ (e.g., in bytes), the weighted cost is:

$$\text{WeightedCost}(u, v) = T(u, v) \times \text{Distance}(P_u, P_v)$$

The total communication cost for an application with a given traffic matrix across all pairs of nodes is the sum of the weighted costs for all active communication pairs:

$$\text{Total Communication Cost} = \sum_{(u,v)\in\text{Pairs}} T(u, v) \times \text{Distance}(P_u, P_v)$$

This total weighted distance provides a quantitative measure of the traffic locality for a given mapping and traffic pattern. A mapping that achieves a lower total weighted distance for a given traffic matrix is considered more locality-aware and is expected to result in better communication performance. Our analysis focuses on comparing this total weighted distance, along with the maximum weighted distance for any single pair, before and after applying the Hilbert curve mapping.

2.4 Distance Metrics

In the context of network topology and communication analysis, distance metrics serve as fundamental tools for quantifying the separation between nodes. The "distance" between two physical locations in a network is not always a simple straight-line distance like in Euclidean geometry; rather, it should reflect the cost or effort required for information to travel between those locations.

Using distance metrics is crucial in analyzing network performance and topology:

- **Quantifying Locality:** Distance metrics provide a quantitative way to assess traffic locality. By calculating the distance between communicating pairs, we can sum or average these values to derive a single number that represents the overall "badness" or "goodness" of a network mapping with respect to communication patterns. A lower total or average distance indicates better locality.

- **Comparing Topologies and Mappings:** Distance metrics allow for objective comparison. We can compare the average distance on different network topologies or, as in this report, compare the distances resulting from different node mapping on the same topology.

2.4.1 Euclidean Distance

The Euclidean distance is perhaps the most intuitive distance metric, representing the straight-line distance between two points in a space. For two points $P_u = (x_u, y_u, z_u)$ and $P_v = (x_v, y_v, z_v)$ in a 3D Cartesian space, the standard Euclidean distance is computed as:

$$D_{Euclidean}(P_u, P_v) = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2 + (z_u - z_v)^2}$$

For a torus network, the Euclidean distance between nodes must account for the wrap-around links. The difference along each dimension is not the simple absolute difference, but the minimum of the direct difference and the difference that wraps around, $\Delta_i = \min(|x_u - x_v|, X - |x_u - x_v|)$, and similarly for Y and Z dimensions. The torus Euclidean distance is then computed using these minimum differences:

$$D^{Torus}_{Euclidean}(P_u, P_v) = \sqrt{\Delta_x^2 + \Delta_y^2 + \Delta_z^2}$$

where $\Delta_x = \min(|x_u - x_v|, X - |x_u - x_v|)$, $\Delta_y = \min(|y_u - y_v|, Y - |y_u - y_v|)$, and $\Delta_z = \min(|z_u - z_z|, Z - |z_u - z_v|)$ for a torus of dimensions $X \times Y \times Z$.

2.4.2 Manhattan Distance

The Manhattan distance, also known as taxicab geometry or L1 distance, is the sum of the absolute differences of the coordinates along each dimension. In a 3D Cartesian space, the standard Manhattan distance between $P_u = (x_u, y_u, z_u)$ and $P_v = (x_v, y_v, z_v)$ is:

$$D_{Manhattan}(P_u, P_v) = |x_u - x_v| + |y_u - y_v| + |z_u - z_v|$$

For a torus network, the Manhattan distance is calculated using the wrap-around adjusted differences $\Delta_x, \Delta_y, \Delta_z$:

$$D^{Torus}_{Manhattan}(P_u, P_v) = \Delta_x + \Delta_y + \Delta_z$$

2.4.3 Chebyshev Distance

The Chebyshev distance, or L∞ distance, is defined as the maximum of the absolute differences between the coordinates along any single dimension. In a 3D Cartesian space, the standard Chebyshev distance between $P_u = (x_u, y_u, z_u)$ and $P_v = (x_v, y_v, z_v)$ is:

$$D_{Chebyshev}(P_u, P_v) = \max(|x_u - x_v|, |y_u - y_v|, |z_u - z_v|)$$

For a torus network, the Chebyshev distance is calculated using the wrap-around adjusted differences $\Delta_x, \Delta_y, \Delta_z$:

$$D^{Torus}_{Chebyshev}(P_u, P_v) = \max(\Delta_x, \Delta_y, \Delta_z)$$

2.5 Node Mapping Strategies

The Hilbert curve mapping strategy leverages the locality-preserving property of the Hilbert curve to assign nodes to torus coordinates. The core idea is to traverse the grid of the torus along a Hilbert curve and assign nodes based on this traversal order.

Specifically, for a torus grid of dimensions $X \times Y \times Z$:

1. Each node location (x, y, z) in the grid is treated as a point in 3D space (or 2D if one dimension is 1).
2. The Hilbert index (a single number) corresponding to each coordinate (x, y, z) is calculated using a Hilbert encoding algorithm. The encoding algorithm ensures that nearby points in the grid tend to have close Hilbert indices.
3. The nodes are then reordered based on their calculated Hilbert indices. The node with the smallest Hilbert index becomes node 0, the node with the next smallest index becomes node 1, and so forth, up to the node with the largest Hilbert index becoming node $N - 1$ (where $N = X \times Y \times Z$).

This Hilbert-based mapping provides an alternative static assignment compared to the baseline canonical mapping, which we evaluate in this report using the traffic-weighted distance metrics detailed in Section 2.3 and Section 2.4. The implementation details of the Hilbert curve encoding and the mapping process are provided in Section 3.1.

3 Methodology

3.1 Hilbert Curve Generation and Mapping Algorithm

3.1.1 Hilbert Value Encoding

The core of the Hilbert curve mapping lies in the encoding algorithm, which translates a multi-dimensional coordinate (e.g., (x, y, z) for a 3D grid node) into a single, unique one-dimensional Hilbert index or key. This index represents the node's position along the Hilbert curve traversing the space. The **encode** function in the **Hilbert_Curve** class implements this transformation.

The algorithm operates on the binary representation of the coordinates and leverages bit manipulation techniques that correspond to the recursive nature of the Hilbert curve construction. For a torus of `num_dims` dimensions, with coordinate values up to $2^{\text{num\_bits}} - 1$ in each dimension, the encoding process for a set of input coordinates (`locs`) proceeds:

```cpp
vector<uint64_t> encode(const vector<vector<uint64_t>>& locs, int num_dims, int num_bits) {
    // Keep the original shape
    int orig_shape = locs.size();

    // Treat the location integers as 64-bit unsigned and then split them up into
    // a sequence of uint8s. Preserve the association by dimension.
    vector<vector<vector<uint8_t>>> locs_uint8(orig_shape, vector<vector<uint8_t>>(num_dims, vector<uint8_t>(8)));
    for (int i = 0; i < orig_shape; i++) {
        for (int j = 0; j < num_dims; j++) {
            for (int byte = 0; byte < 8; byte++) {
                locs_uint8[i][j][byte] = (uint8_t)((locs[i][j] >> (8 * (7 - byte))) & 0xFF);
            }
        }
    }

    // Turn into bits and truncate to num_bits
    vector<vector<vector<bool>>> gray(orig_shape, vector<vector<bool>>(num_dims));
    for (int i = 0; i < orig_shape; i++) {
        for (int j = 0; j < num_dims; j++) {
            gray[i][j] = unpack_and_truncate(locs_uint8[i][j], num_bits);
        }
    }

    // Encoding process (dim-wise bit manipulation)
    for(int bit = 0; bit < num_bits; bit++) {
        for (int dim = 0; dim < num_dims; dim++) {
            for (int idx = 0; idx < orig_shape; idx++) {
                bool mask = gray[idx][dim][bit];

                if (mask) {
                    for (int b = bit + 1; b < num_bits; b++) {
                        gray[idx][0][b] = !gray[idx][0][b];
                    }
                } else {
                    for (int b = bit + 1; b < num_bits; b++) {
                        bool diff = gray[idx][0][b] ^ gray[idx][dim][b];
                        if (diff) {
                            gray[idx][0][b]   = !gray[idx][0][b];
                            gray[idx][dim][b] = !gray[idx][dim][b];
                        }
                    }
                }
            }
        }
    }

    // Flatten and convert to binary
    vector<vector<bool>> flat_gray = flatten_gray(gray, num_dims, num_bits);
    vector<vector<bool>> hh_bin = gray2binary(flat_gray);

    // Pad back to 64 bits
    unsigned int extra_dims = 64 - num_bits * num_dims;
    vector<vector<bool>> padded;
    padded.reserve(hh_bin.size());

    for (const auto& row : hh_bin) {
        vector<bool> paddedRow(64, false);
        for (size_t i = 0; i < row.size(); i++) {
            paddedRow[extra_dims + i] = row[i];
        }
        padded.push_back(paddedRow);
    }

    vector<uint64_t> result = convertAll(padded);
    return result;
}
```

Listing 1: Hilbert Curve Encoding Algorithm (`Hilbert_Curve::encode`)

The encoding process, as implemented, follows these conceptual steps:

1. **Initial Bit Extraction and Truncation:** The input coordinates (`locs`), provided as `uint64_t`, are first converted into their byte representation and then unpacked into boolean vectors representing individual bits. Only the `num_bits` relevant to the dimensions of the torus are retained for each coordinate. This prepares the data for bit-level manipulation.
2. **Core Bit Transformation Loop:** This is the most complex part of the algorithm, involving nested loops that iterate through each bit level (`bit`) and each dimension (`dim`) for every point (`idx`). Inside these loops, a set of bitwise operations (XOR and NOT) are applied to the *lower* bits (`b > bit`) of the dimensions. These operations are derived from the geometric transformations (rotations and reflections) required at each stage of the recursive Hilbert curve construction. The logic essentially reorders and transforms the bits based on the higher-order bits to achieve the locality-preserving mapping.
3. **Flattening and Gray-to-Binary Conversion:** After the bit transformations are complete, the bits for each point (which are still conceptually separated by dimension and bit level) are flattened into a single sequence. This sequence is generated such that bits from the same bit level across all dimensions are grouped together (`b * num_dims + d`), effectively interleaving the bits from the original coordinates. The resulting interleaved bit string is in Gray code format. A subsequent step converts this Gray code representation into its standard binary form.
4. **Padding and Final Output:** The final binary bit sequence, which is `num_dims` × `num_bits` long, represents the unique Hilbert index for the original coordinate. This sequence is then padded with leading zero bits to a full 64-bit length and packed into a `uint64_t` integer. The function returns a vector of these 64-bit Hilbert indices, one for each input coordinate.

Helper functions like `unpack_and_truncate`, `flatten_gray`, `gray2binary`, and `convertAll` handle the conversions between different bit and integer representations throughout the process, facilitating the bit manipulations at the core of the algorithm. The output vector of Hilbert indices defines the new linear order for the nodes, which is then used in the mapping analysis.

3.1.2 Node Reordering based on Hilbert Index

Once the Hilbert index has been computed for each original node coordinate using the encoding algorithm described in Section 3.1.1, these indices are used to define a new, locality-aware ordering of the nodes. This reordering forms the basis of the Hilbert curve mapping strategy.

The process involves the following steps:

1. **Pairing Index with Original Coordinate:** For each node located at original coordinates (x, y, z), its computed Hilbert index h is paired with its original coordinate tuple, forming a set of pairs $(h, (x, y, z))$. The total number of such pairs is equal to the total number of nodes in the torus grid ($N = X \times Y \times Z$).
2. **Sorting by Hilbert Index:** These pairs are then sorted in ascending order based on the Hilbert index h. This sorting arranges the original physical nodes according to their sequence along the one-dimensional Hilbert curve traversal of the grid. The point with the smallest Hilbert index will be first in the sorted list, the point with the next smallest index will be second, and so on. This step is implemented using a standard sorting algorithm with a custom comparison function that compares the first element of each pair (the Hilbert index),.
3. **Creating the New Coordinate Mapping:** The sorted list implicitly defines the new mapping. The **position** of a pair in the sorted list corresponds to the position in the network, and the **original coordinate** associated with that pair is the node location.

This process effectively reorders the original multi-dimensional grid points based on their Hilbert indices, creating a linear sequence that preserves much of the spatial locality. This new mapping (`newCoords`) is then used in the analysis functions to calculate distances between communicating pairs in the Hilbert-ordered space.

3.1.3 Calculation of Traffic-Weighted Distances

To evaluate the effectiveness of a node mapping strategy in preserving traffic locality, we calculate the **traffic-weighted distance** for the communication patterns defined by the input traffic matrix. This metric accounts for both the separation between communicating nodes and the volume of data exchanged between them. A lower traffic-weighted distance indicates better locality, as high-traffic communication is concentrated between physically nearby nodes.

The calculation process for a given mapping and a given traffic matrix T, which specifies the traffic volume $T(u,v)$ for each pair of communicating original logical nodes (u,v), proceeds as follows:

1. **Traffic Aggregation:** The raw traffic data is processed to obtain the total traffic volume $T(u,v)$ for every unique ordered pair of original logical node IDs (u,v) that communicate. If the data specifies traffic in both directions (u to v and v to u), these can be treated as separate pairs or potentially combined depending on the analysis requirements.

2. **Coordinate Mapping:** For each communicating pair (u,v), their physical coordinates under the current mapping must be determined.

   - If evaluating the **baseline canonical mapping**, the physical coordinates P_u and P_v are simply the original grid coordinates (x_u, y_u, z_u) and (x_v, y_v, z_v) corresponding to the IDs u and v in the default grid ordering.
   - If evaluating a **reordered mapping** (such as the Hilbert mapping), the physical coordinates P_u and P_v are the original grid coordinates assigned to the IDs u and v after the mapping transformation (e.g., the coordinates stored at the u-th and v-th positions in the Hilbert-ordered list of points, as derived in Section 3.1.2).

3. **Distance Calculation:** For each communicating pair (u,v), the distance between their corresponding coordinates P_u and P_v is calculated using the torus-aware distance metrics defined in Section 2.4. This is done independently for the Euclidean ($D^{Torus}_{Euclidean}$), Manhattan ($D^{Torus}_{Manhattan}$), and Chebyshev ($D^{Torus}_{Chebyshev}$) metrics.

4. **Weighted Distance Computation:** The traffic-weighted distance for the pair (u,v) for a given metric is computed by multiplying the traffic volume by the calculated physical distance:

$$\text{Weighted Distance}(u, v, \text{Metric}) = T(u,v) \times D^{Torus}_{\text{Metric}}(P_u, P_v)$$

   This step is performed for every communicating pair and for each of the three distance metrics.

5. **Aggregation for Summary Metrics:** The individual pair-wise weighted distances are aggregated to derive summary metrics that characterize the overall locality of the mapping for the given traffic pattern:

   - **Total Traffic-Weighted Distance:** The sum of the weighted distances across all communicating pairs for a given metric:

$$\text{Total Weighted Distance (Metric)} = \sum_{(u,v)\in\text{Pairs}} \text{Weighted Distance}(u, v, \text{Metric})$$

     This metric represents the cost of all communication under the given mapping.

   - **Maximum Traffic-Weighted Distance:** The single largest weighted distance among all communicating pairs for a given metric:

$$\text{Maximum Weighted Distance (Metric)} = \max_{(u,v)\in\text{Pairs}} \text{Weighted Distance}(u, v, \text{Metric})$$

     This metric highlights the single most ”expensive” communication pair in terms of combined volume and distance.

These total and maximum traffic-weighted distances, computed for each distance metric under different mappings, provide the data used to compare the effectiveness of mapping strategies in improving traffic locality. The C++ implementation automates these calculations based on the parsed traffic data and the generated mapping information.

4 Experiments and Results

4.1 Visual Results

4.1.1 Torus Grid Visualization

Figure Figure 11 illustrates the grid structure for the $2 \times 2 \times 4$ configuration, and Figure Figure 12 shows the larger $2 \times 8 \times 16$ configuration.

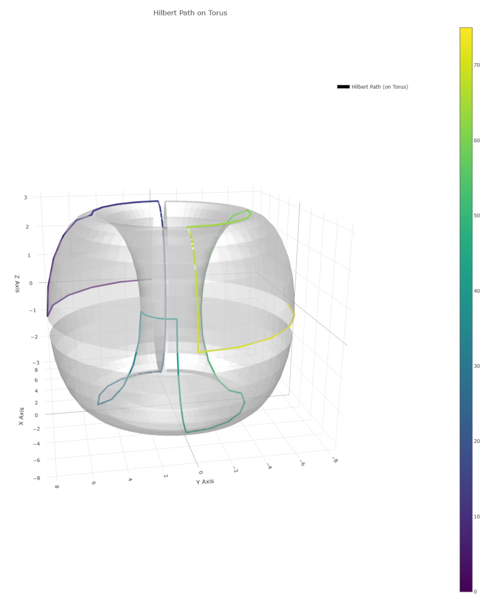


Figure 11: Visualization of the $2 \times 2 \times 4$ Torus grid structure.

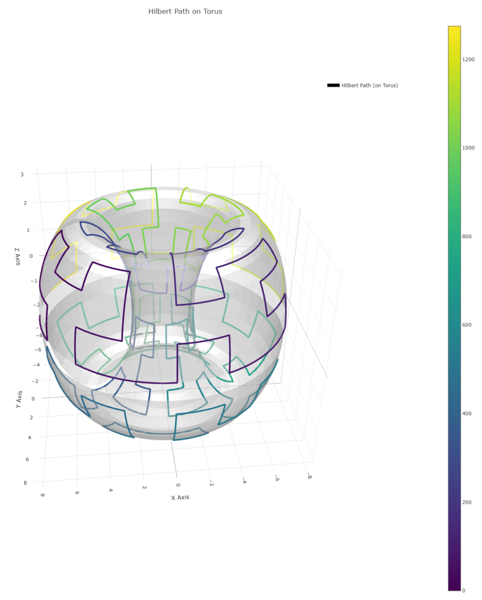


Figure 12: Visualization of the $2 \times 8 \times 16$ Torus grid structure.

4.1.2 Hilbert Curve Path Visualization

The Hilbert curve mapping reorders the nodes based on their traversal along a path that fills the grid. Visualizing this path on the original grid provides insight into how locality is preserved. Figure Figure 13 shows the Hilbert curve path through the $2 \times 2 \times 4$ grid, and Figure Figure 14 shows the path through the $2 \times 8 \times 16$ grid. The color scale indicates the sequential order along the Hilbert curve.

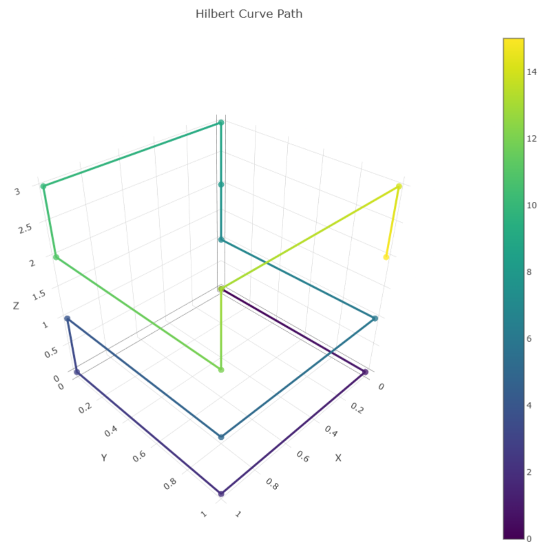


Figure 13: Visualization of the Hilbert curve path through the $2 \times 2 \times 4$ grid. Color represents sequence along the curve.

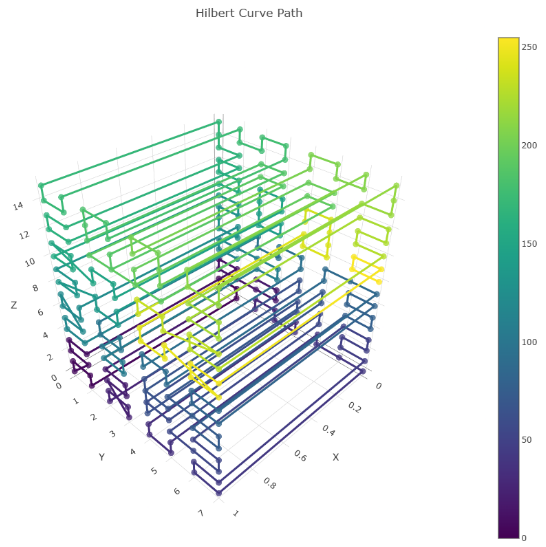


Figure 14: Visualization of the Hilbert curve path through the $2 \times 8 \times 16$ grid. Color represents sequence along the curve.

These figures visually demonstrate how the Hilbert curve attempts to linearize the multi-dimensional grid while keeping spatially close points (represented by adjacent colors) near each other in the 1D sequence.

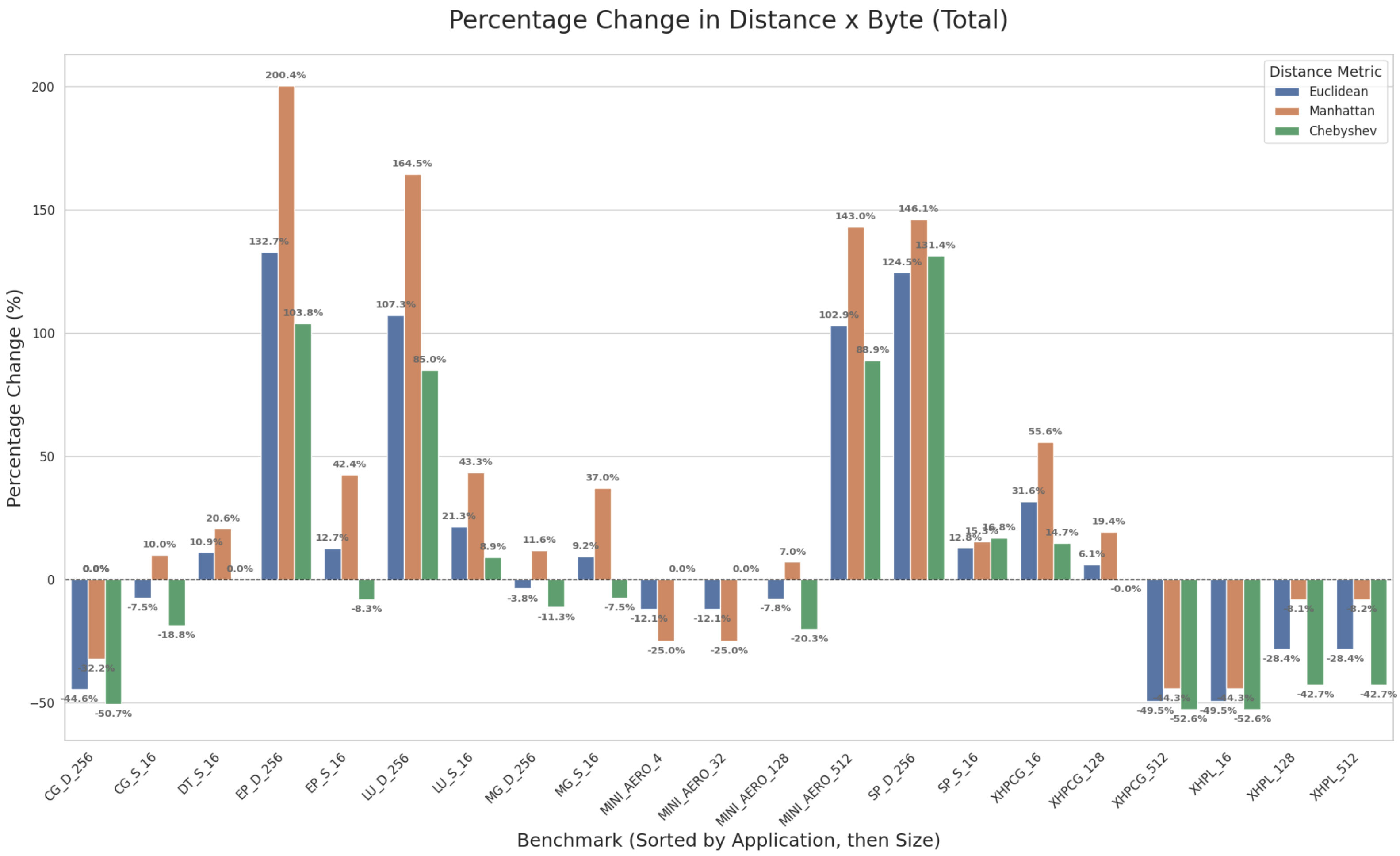
4.2 Impact of Hilbert Mapping

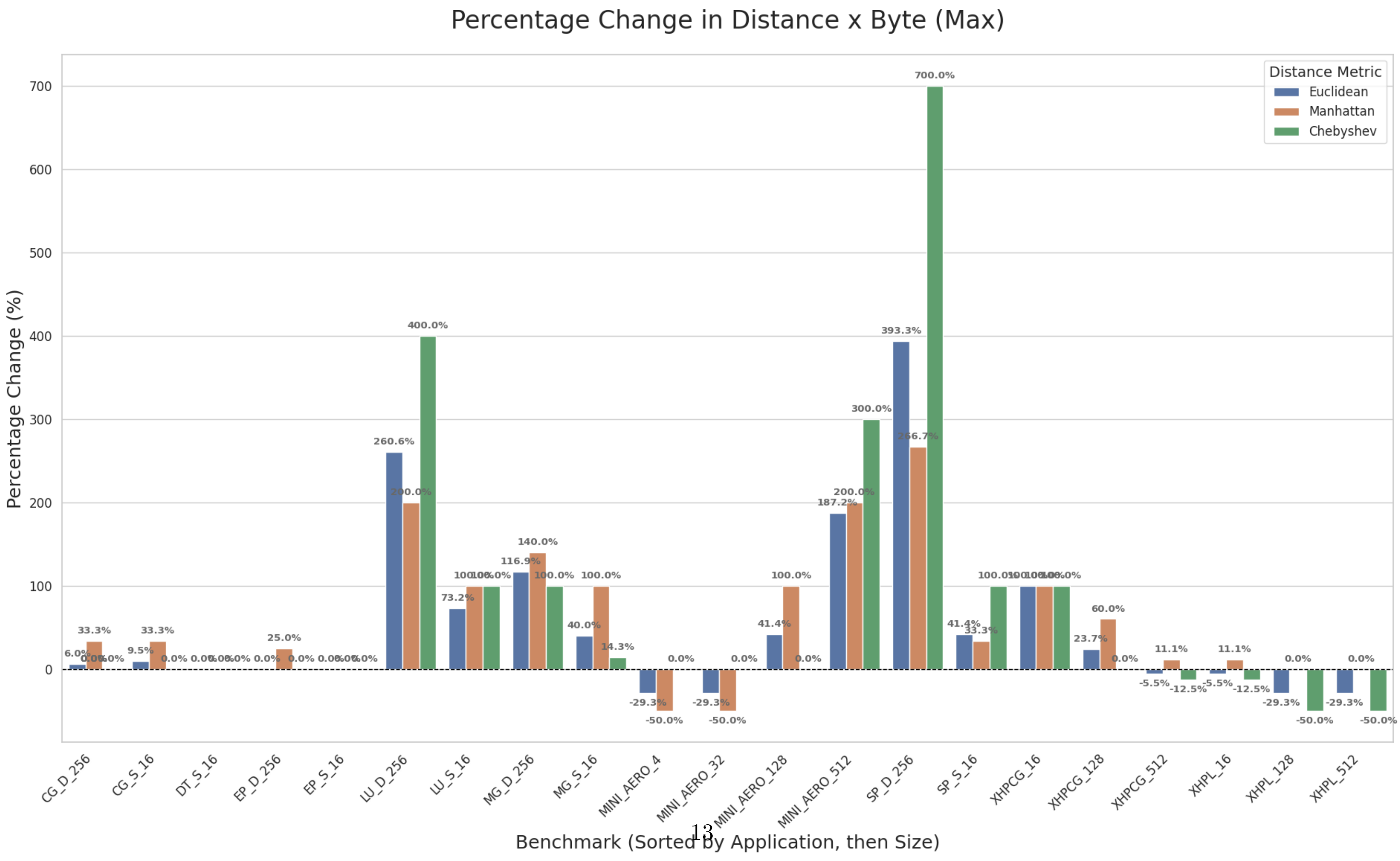
4.2.1 Percentage Change in Weighted Distances

The impact of the Hilbert mapping is quantified by the percentage change in total and maximum traffic-weighted distances compared to the baseline canonical mapping. A negative percentage indicates a reduction in weighted distance (improvement in locality), while a positive percentage indicates an increase (worsening locality).

The percentage change ($\Delta\%$) for a metric M is calculated as:

$$\Delta\%(M) = \frac{M_{Hilbert} - M_{Baseline}}{M_{Baseline}} \times 100$$


(a) Comparison of Traffic-Weighted Distances (Total)


(b) Comparison of Traffic-Weighted Distances (Max)

- **"CG" Traffic Patterns (Traffic_cg_s_16.csv, Traffic_cg_d_256.csv):** For these patterns, the Hilbert curve mapping consistently resulted in a significant reduction in total traffic-weighted distance across all distance metrics, as shown in Figure 15a (e.g., -44.57% Euclidean, -32.19% Manhattan, -50.74% Chebyshev for the $2 \times 8 \times 16$ torus). This suggests that the "CG" patterns—along with other similar communication patterns such as **MiniAero**, **XHPCG**, and **XHPL**—when remapped according to the Hilbert curve, exhibit substantially improved traffic locality. This improvement is likely due to these patterns having communication structures where frequently communicating pairs, although potentially not nearest-neighbors in the original grid, exhibit relationships that are well-preserved when the grid is linearized by the Hilbert curve. The Hilbert curve's ability to keep generally nearby points close in the 1D sequence appears effective in reducing the distance for these communication patterns.
- **"SP" Traffic Patterns (Traffic_sp_s_16.csv, Traffic_sp_d_256.csv):** In contrast, the "SP" patterns showed either a slight worsening or, for the larger $2 \times 8 \times 16$ torus (Traffic_sp_d_256.csv), a dramatic increase in both total and maximum traffic-weighted distances after applying the Hilbert mapping (shown in Figure 15). This indicates that for these patterns—as well as for similar patterns such as **DT**, **EP**, **LU**, **MG**, and **SP**—the Hilbert reordering is detrimental to traffic locality.

This difference in behavior can be attributed to the nature of the "SP" traffic patterns and their interaction with the torus topology's wrap-around links under different mappings. While "CG" patterns might have more localized communication (potentially along diagonals that align well with Hilbert's linearization), the "SP" patterns appear to involve communication between nodes that are more dispersed across the grid. Crucially, the "SP" communication seems to frequently occur between nodes that, while physically distant in a simple rectilinear grid sense, benefit significantly from the **wrap-around links** provided by the torus topology in the baseline mapping. The standard torus distance calculation for these pairs leverages the shortest path, which might utilize a wrap-around link, resulting in a relatively low distance contribution to the baseline total/maximum.

When the nodes are reordered by the Hilbert curve, the spatial relationships are linearized. However, communicating pairs in the "SP" pattern that heavily relied on those specific long-distance wrap-around "shortcuts" in the baseline mapping may now be mapped to positions in the Hilbert sequence such that their **original** coordinates, when evaluated with torus distance, no longer exhibit that favorable low-distance property relative to other pairs. The Hilbert curve preserves general locality, but it may disrupt specific long-distance relationships that were effectively shortened by the torus wrap-around in the original grid layout, leading to a substantial increase in weighted distance for these patterns.

Furthermore, the negative impact on "SP" patterns is significantly more pronounced for the larger $2 \times 8 \times 16$ torus compared to the $2 \times 2 \times 4$ torus. This suggests that the scale and possibly the aspect ratio of the torus, combined with the dispersed nature of the "SP" traffic pattern, amplify the detrimental effect of the Hilbert reordering on the utilization of torus wrap-around benefits.

5 Conclusion

This report investigated the effectiveness of using Hilbert space-filling curves as a static mapping strategy to improve traffic locality on torus network. By developing a custom analysis tool, we evaluated the impact of Hilbert reordering on communication costs using traffic-weighted distance metrics.

5.1 Summary of Findings

The primary goal of this research was to assess whether mapping application nodes onto physical torus coordinates according to a Hilbert curve traversal could reduce communication cost compared to a simple canonical grid mapping. We developed a C++ tool that simulates torus topologies, processes arbitrary traffic matrices, calculates Hilbert indices using a bit-manipulation algorithm, and compares traffic-weighted Euclidean, Manhattan, and Chebyshev distances before and after applying the Hilbert-based reordering.

Our experiments, conducted on $2 \times 2 \times 4$ and $2 \times 8 \times 16$ torus configurations with different synthetic traffic patterns , with varying results regarding the Hilbert mapping's effectiveness.

5.2 Analysis of Results

The analysis of traffic-weighted distances revealed a significant dependency on the characteristics of the traffic pattern being mapped.

- For the "CG" traffic patterns, the Hilbert curve mapping consistently resulted in a notable reduction across all evaluated weighted distance metrics (e.g., up to -44.57% for Total Euclidean distance on the $2 \times 8 \times 16$

torus). This indicates that the "CG" patterns' inherent communication structure, involving diagonal-like dependencies, aligns well with the locality-preserving properties of the Hilbert curve's linearization, leading to improved traffic locality. This trend is also observed in other patterns with similar characteristics such as **MiniAero**, **XHPCG**, and **XHPL**, which show comparable reductions in traffic-weighted distances.

- In contrast, the "SP" traffic patterns generally showed an increase in weighted distances after Hilbert mapping. For the larger $2 \times 8 \times 16$ torus with the "SP" pattern, this increase was dramatic (e.g., over 100% increase in Total Weighted Distance and several hundred percent increase in Maximum Weighted Distance). This detrimental effect is attributed to the "SP" patterns involving more dispersed communication that likely benefits significantly from the torus's wrap-around shortcuts in the baseline grid mapping. The Hilbert reordering, while preserving general locality, appears to disrupt these specific long-distance relationships that are favorably mapped by the wrap-around links in the default layout. Similar behavior is seen in other patterns such as **DT**, **EP**, **LU**, and **MG**, which also experience degraded traffic locality under Hilbert mapping.

These findings highlight that the Hilbert curve mapping is not a universal solution for all traffic patterns on torus networks. Its effectiveness is contingent on how well the application's communication graph's spatial characteristics align with the Hilbert curve's traversal path. The Manhattan distance metric often showed particularly relevant changes, aligning with its interpretation as hop count in grid networks.

5.3 Future Work

Based on the findings and of this research, several directions for future work are suggested:

- **Evaluate Other Mapping Strategies:** Compare Hilbert mapping against other static techniques and explore dynamic mapping approaches for torus networks.
- **Explore Other Space-Filling Curves:** Investigate the effectiveness of other space-filling curves (e.g., Z-order/Morton, Peano) for torus mapping and compare their locality properties.
- **Integrate with Network Simulators:** Couple the mapping framework with a detailed network simulator to evaluate the impact of Hilbert mapping on actual network performance metrics like latency, throughput, and congestion under realistic traffic conditions and routing protocols.
- **Investigate Adaptive Approaches:** Explore hybrid strategies that combine Hilbert mapping with local dynamic remapping or routing adjustments.
- **Hardware Considerations:** Consider the practical implications and potential hardware overheads of implementing Hilbert-based mapping or reordering in a real system.

References

[1] N. R. Agida *et al.*, "Blue Gene/L Torus Interconnection Network," *IBM Journal of Research and Development*, vol. 49, no. 2/3, pp. 265–280, 2005. DOI: 10.1147/rd.492.0265.

[2] G. Peano, "Sur une courbe, qui remplit toute une aire plane," *Mathematische Annalen*, vol. 36, no. 1, pp. 157–160, 1890. DOI: 10.1007/BF01199438.

[3] D. Hilbert, "Über die stetige Abbildung einer Linie auf ein Flächenstück," *Mathematische Annalen*, vol. 38, pp. 459–460, 1891. DOI: 10.1007/BF01199431.

[4] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the Hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 124–141, 2001. DOI: 10.1109/69.908985.