A Novel Construction Method for \( n \)-Dimensional Hilbert Space-Filling Curves*

Chih-Sheng CHEN††, Student Member, Shen-Yi LIN†, Min-Hsuan FAN†, and Chua-Huang HUANG†, Nonmembers

SUMMARY We develop a novel construction method for \( n \)-dimensional Hilbert space-filling curves. The construction method includes four steps: block allocation, Gray permutation, coordinate transformation and recursive construction. We use the tensor product theory to formulate the method. An \( n \)-dimensional Hilbert space-filling curve of \( 2^n \) elements on each dimension is specified as a permutation which rearranges \( 2^n \) data elements stored in the row major order as in C language or the column major order as in FORTRAN language to the order of traversing an \( n \)-dimensional Hilbert space-filling curve. The tensor product formulation of \( n \)-dimensional Hilbert space-filling curves uses stride permutation, reverse permutation, and Gray permutation. We present both recursive and iterative tensor product formulas of \( n \)-dimensional Hilbert space-filling curves. The tensor product formulas are directly translated into computer programs which can be used in various applications. The process of program generation is explained in the paper.

key words: block recursive algorithm, tensor product, \( n \)-dimensional Hilbert space-filling curve, Gray permutation

1. Introduction

A space-filling curve is used to express the locality of multidimensional data in a one-dimensional space and to traverse on a multi-dimensional object exactly once and without crossing the path [27]. In 1891, D. Hilbert presented a geometric interpretation of traversing two-dimensional \( 2 \times 2 \) space-filling curves [7]. The order can be used to arrange data elements in various applications such as image pixel allocation [1], [23], [24], VLSI component layout [26], [30], and R-tree indexing [5], [14], [15], [17], [18] to increase locality efficiency. Recently, 3-D space-filling curves are used to detect and classify functional Magnetic Resonance Imaging (fMRI) activation patterns of clinical medical images [16]. The 3-D Hilbert wire antenna can increase the wire length and reduce the resonant frequency with the antenna occupying the same volume [29].

Suppose the data elements on an \( n \)-dimensional cube are initially stored in the row major order as in C language or the column major order as in FORTRAN language. The Hilbert space-filling curve is viewed as a permutation function of a block recursive structure that rearranges the data elements to the order of traversing an \( n \)-dimensional Hilbert space-filling curve.

Tensor product, also known as Kronecker product [6], has been used to design and implement block recursive algorithms such as fast Fourier transform [10], [11], Strassen’s matrix multiplication [8], parallel prefix algorithms [4], 2-dimensional Hilbert space-filling curve [20], Karatsuba’s multiplier [21]. For different architecture characteristics, such as vector processors, parallel multiprocessors, and distributed-memory multiprocessors, tensor product formulas can be manipulated using appropriate algebraic theorems and then translated to high-performance programs. Tensor product formulas can also be used to specify data allocation and generate efficient programs for multi-level memory hierarchy including cache memory, local memory, and external memory.

In this paper, we develop a construction method for \( n \)-dimensional Hilbert space-filling curves and using the tensor product theory to formulate the method. We express the Hilbert permutation as an algebraic formula consisting of tensor product, direct sum, matrix product operations and some specific permutations: stride permutation, reverse permutation, and Gray permutation. These operations and permutations can be mapped to various statements of high-level programming languages. Hence, a tensor product formula of \( n \)-dimensional Hilbert space-filling curves can be directly translated into a computer program. The program rearranges data elements of an \( n \)-dimensional array from the row (or column) major order to the Hilbert space-filling curve order.

The paper is organized as the following. Related works of Hilbert space-filling curves are given in Sect. 2. In Sect. 3, we briefly explain the algebraic theory of tensor product and other related operations. We develop a method to construct an \( n \)-dimensional Hilbert space-filling and using tensor product to formulate the method in Sect. 4. In addition, we use a 3-D Hilbert space-filling curve to describe the construction method and using an example to explain how to translate a 4-D row-major order point to a Hilbert space-filling curve point. Program generation from tensor product formulas of 3-D Hilbert space-filling curves is explained in Sect. 5. Concluding remarks and future works are given in Sect. 6.

2. Related Works

Since D. Hilbert presented the Hilbert space-filling curve
in 1981 [7], there have been several research works on how to formally specify it using either an operational model or a functional model. The Hilbert space-filling curve has been viewed as a one-to-one mapping function by Butz [2], [3]. He proposed an algorithm to compute the mapping function with bit operations such as exclusive OR, shifting, etc. Jagadish had analyzed the clustering properties of the Hilbert space-filling curve [9]. He showed that the Hilbert space-filling curve achieves the best clustering, i.e., the best space-filling curve in minimizing the number of clusters. Moon et al. provided closed-form formulas of the number of clusters required by a given query region of an arbitrary shape for the Hilbert space-filling curve [25]. Kamata et al. proposed a nonrecursive algorithm for the n-dimensional Hilbert space-filling curve using look-up tables [12], [13]. Liu and Schrack presented an algebraic algorithm of encoding and decoding of the 3-dimensional Hilbert order using bit operations [22]. A mathematical history of the Hilbert space-filling curve was presented by Sagan [31]. Li et al. based on a static evolvement rule table and proposed algorithms for generating an n-dimensional Hilbert space-filling curve [19].

3. Overview of Tensor Product Operations

In this section, we give an brief overview of the algebraic operations and the properties used in formulating the n-dimensional Hilbert space-filling curve permutation. The operations explained include tensor product, direct sum, vector reversal and stride permutation.

Tensor product is a matrix operation which builds a “large” matrix from two “small” matrices. It is defined as below:

**Definition 3.1 (Tensor Product)** Let $A$ and $B$ be two matrices of size $m \times n$ and $p \times q$, respectively. The tensor product of $A$ and $B$ is the block matrix obtained by replacing each element $a_{ij}$ by $a_{i,j}B$, i.e., $A \otimes B$ is an $mp \times nq$ matrix defined as

\[
A \otimes B = \begin{bmatrix}
a_{0,0}B_{p0q} & \cdots & a_{0,n-1}B_{p0q} \\
\vdots & \ddots & \vdots \\
a_{m-1,0}B_{p0q} & \cdots & a_{m-1,n-1}B_{p0q}
\end{bmatrix}.
\]

Let $F_m^n$ be the vector space of $n$-tuples over field $F$ and let $F_m^{mn}$ be the vector space of $m \times n$ matrices. The collection of elements $\{e_i^m | 0 \leq i < m\}$, where $e_i^m$ is the vector with a one in the $i$-th position and zeros elsewhere, form the standard basis for $F_m^n$.

**Definition 3.2 (Tensor Bases)** Let $F^n$ be the vector space of $n$-tuples over field $F$, a collection of elements $\{e_i^n \otimes e_j^n \otimes \cdots \otimes e_k^n | 0 \leq i_1 < n_1, 0 \leq i_2 < n_2, \cdots, 0 \leq i_k < n_k\}$ are called a tensor bases of $F^n \otimes F^n \otimes \cdots \otimes F^n$.

A tensor basis can be linearized (or factorized) as below:

\[
e_{i_1}^{n_1} \otimes e_{i_2}^{n_2} \otimes \cdots \otimes e_{i_k}^{n_k} = e_{i_1 \oplus i_2 \oplus \cdots \oplus i_k}^{n_1 + n_2 + \cdots + n_k}.
\]

Another operation that builds a “large” matrix from two “small” matrices is direct sum operation.

**Definition 3.3 (Direct Sum)** Let $A$ and $B$ be two matrices $m \times n$ and $p \times q$, respectively. The direct sum of $A$ and $B$ is an $(m + p) \times (n + q)$ matrix defined as

\[
A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}.
\]

Three permutations are used in formulating the 3-dimensional Hilbert space-filling curve permutation. They are stride permutation, reverse permutation, and Gray permutation.

**Definition 3.4 (Stride Permutation)** A stride permutation $L_{mn}^{pq}$ is defined by

\[
L_{mn}^{pq}(e_i^m \otimes e_j^n) = e_{i+q}^m \otimes e_{j+p}^n.
\]

$L_{mn}^{pq}(e_i^m \otimes e_j^n)$ is referred to as the stride permutation which permutes the tensor product of two vector bases. If an $m \times n$ matrix is stored in the column major order, its basis is isomorphic to $e_0^m \otimes e_0^n$. Stride permutation is exactly the transposition operation transforming the matrix from the column major ordering allocation to the row major ordering. Stride permutation also corresponds to exchange of the coordinate system.

**Definition 3.5 (Reverse Permutation)** A reverse permutation $J_n$ is defined by

\[
J_n e_i^n = e_{(n-1)-i}^n.
\]

$J_n$ map the basis element $e_i^n$ to the basis element $e_{(n-1)-i}^n$. Reverse permutation corresponds to reversal of the coordinate system. It is also used in the definition of Gray permutation.

**Definition 3.6 (Gray Permutation)** The $n$-bit Gray permutation $G_2^n$ is defined as

\[
G_2 = I_2, \quad G_2^n = G_2^{n-1} \oplus G_2^{n-1} \circ (I_2 \otimes G_2^{n-1}).
\]

The permutation matrix and inverse permutation matrix of Gray permutation $G_2^n$ are:

\[
G_2^n = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix},
\]

\[
G_2^n = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.
\]
There are some properties of tensor products, direct sums, stride permutations, and reverse permutations used in this paper. The readers are referred to [20] for these properties.

4. Construction of \( n \)-Dimensional Hilbert Space-Filling Curves

An \( n \)-dimensional Hilbert space-filling curve \( H^n \) can be viewed as a hypecube consisting of \( 2^n \) subcubes such that each subcube is of size \( 2^{n(r-1)} \). Recursively, each subcube is an \( n \)-dimensional Hilbert space-filling curve \( H^n_{r-1} \). The subcubes are connected in the Gray permutation order. Gray permutation is an ordering of non-negative integers where two successive values differ in only one bit. Hence, an \( H^n_r \) Hilbert space-filling curve is recursively constructed from \( H^n_{r-1} \) Hilbert space-filling curves. For example, a \( 2 \times 2 \times 2 \), \( H^3 \), and a \( 4 \times 4 \times 4 \), \( H^3 \), Hilbert space-filling curve are shown in Fig. 1. \( H^2_x \) is a curve connecting eight copies of \( H^1_x \) in different orientations.

Suppose the nodes of the hypercube represent some data elements of an application. The collection of these data elements must be stored in computer memory according to a given order, usually, the row-major or the column-major order. These location orders are natural, but they are lack of locality efficiency. If the data elements are stored in an \( n \)-dimensional Hilbert space-filling curve order, it may improve locality access and spatial structure. In this paper, we develop a method to construct the \( n \)-dimensional Hilbert space-filling curves for rearranging computer data in the row-major order into the \( n \)-dimensional Hilbert space-filling curve order. The construction method for \( n \)-dimensional Hilbert space-filling curves includes four steps: (1) block allocation, (2) Gray permutation, (3) coordinate transformation, and (4) recursive construction. We will express formal specification of these steps using tensor product formulas in the following subsections. Suppose the \( 2^{nr} \) elements are initially stored in the row major order. The index of point \((x_0, x_1, \ldots, x_j, \ldots, x_{n-1})\), where \( 0 \leq x_j < 2^r \), is described by tensor basis \( e_{x_0}^r \otimes e_{x_1}^r \otimes \cdots \otimes e_{x_{n-1}}^r \).

4.1 Block Allocation

The purpose of block allocation is to reallocate the initial row-major ordering data to \( 2^n \) blocks of \( 2^{n(r-1)} \) elements and these \( 2^n \) blocks become a hypercube structure. Note that the subcubes and the elements in each subcube are allocated in the row-major order. We can use the \( B^n_r \), a tensor product formulas, \( \prod_{i=0}^{n-2} (I_{2^{r+1}} \otimes L_{2^{r+1}}^1 \otimes I_{2^{n(r-1)+1}}) \) to describe this operation.

Applying \( B^n_r \) to the initial row major ordering tensor basis, we obtain the following basis:

\[
\left( I_{2^{r+1}} \otimes L_{2^{r+1}}^1 \otimes I_{2^{n(r-1)+1}} \right)
\]

Therefore, Suppose the nodes of the hypercube represent some data elements of an application.

4.2 Gray Permutation

Gray permutation, \( G^n_r \), let the \( 2^n \) subcubes follow a Gray code order but the elements in each subcube are also allocated in the row-major order. The \( G^n_r \) can be decomposed to \( G^n_r \otimes I_{2^{n(r-1)}} \).

Applying the Gray permutation to the resulting tensor basis of block allocation, we obtain:

\[
G^n_r = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

Fig. 1 3-dimensional Hilbert space-filling curves.
4.3 Coordinate Transformation

After the previous step, these subcubes are connected in Gray permutation order, but the ending point of a subcube and the starting point of following subcube may not be connected adjacenty. We will transform the coordinate of subcubes and connect each other. The beginning of the procedure is to search the starting and the ending points of these subcubes. Afterward, we operate XOR between the these points and use the stride and reverse permutations to generate the coordinate transformation for each subcube.

In the orginal coordinate system, if the ending point of a subcube is \((\hat{b}_0x'_0, \hat{b}_1x'_1, \cdots, \hat{b}_{n-1}x'_{n-1})\), the starting point of the following subcube must be \((\hat{b}_0x'_0, \hat{b}_1x'_1, \cdots, \hat{b}_{j-1}x'_{j-1}, \hat{b}_jx'_j, \cdots, \hat{b}_{n-1}x'_{n-1})\), for some j, \(0 \leq j < n\). Also, the Hamming distance of the starting and ending points of a subcube is 1. Hence, we will develop the subcube connection algorithm to determine the starting and the ending points of these subcubes.

We use two unary operations \(\downarrow\) and \(\updownarrow\) to extract subcube index and subcube point coordinate, respectively, and a binary operation \(\oplus\) to restore point coordinate as the following:

\[
\downarrow((\hat{b}_0x'_0, \hat{b}_1x'_1, \cdots, \hat{b}_jx'_j, \cdots, \hat{b}_{n-1}x'_{n-1})) = (\hat{b}_0, \hat{b}_1, \cdots, \hat{b}_j, \cdots, \hat{b}_{n-1}),
\]

\[
\updownarrow((\hat{b}_0x'_0, \hat{b}_1x'_1, \cdots, \hat{b}_jx'_j, \cdots, \hat{b}_{n-1}x'_{n-1})) = (x'_0, x'_1, \cdots, x'_j, \cdots, x'_{n-1}),
\]

\[
(\hat{b}_0, \hat{b}_1, \cdots, \hat{b}_j, \cdots, \hat{b}_{n-1}) \oplus (x'_0, x'_1, \cdots, x'_j, \cdots, x'_{n-1}) = (\hat{b}_0x'_0, \hat{b}_1x'_1, \cdots, \hat{b}_jx'_j, \cdots, \hat{b}_{n-1}x'_{n-1}).
\]

For example, \(\downarrow((00, 00, 10)) = (0, 0, 1)\), \(\updownarrow((01, 00, 10)) = (1, 0, 0)\) and \(\updownarrow((01, 00, 10)) = (01, 00, 10)\).

The construction method for n-dimensional Hilbert space-filling curves is a recursive method. It is sufficient to use the n-dimensional space of size \(2^n\) on each dimension, i.e., the Hilbert space-filling curve of \(2^{2n}\) points, to determine the coordinate transformation.

The subcube connection algorithm includes three steps. We use the \(4 \times 4 \times 4\) Hilbert space-filling curve to explain the procedure and use bit representation of the coordinate of a point, \((\hat{b}_0x'_0, \hat{b}_1x'_1, \hat{b}_2x'_2)\). Recall that the higher bit is remarked with a hat, \(\hat{b}_i\), to highlight a subcube index bit. We begin from the point \((00, 00, 00)\) in the orginal coordinate system.

We apply the extraction operations to obtain subcube index and subcube point coordinate of \((00, 00, 00)\), i.e., \(\downarrow((00, 00, 00)) = (0, 0, 0)\) and \(\updownarrow((00, 00, 00)) = (0, 0, 0)\). The subcube point coordinate of the ending point has Hamming distance 1 from \((0, 0, 0)\), i.e., \((0, 0, 1)\), \((1, 0, 0)\), or \((0, 1, 0)\). Namely, the coordinate of the ending point in the orginal space is \((00, 00, 01)\), \((00, 01, 00)\), or \((01, 00, 00)\).

The subcubes are stored in the order of Gray permutation. Hence, the index of the subcube following \((0, 0, 0)\) is \((0, 0, 1)\). The coordinate of the starting point of the next subcube is \((0, 0, 1)\) \(\oplus (0, 0, 0) = (00, 00, 10)\). This starting point has Euclidean distance 1 from the ending point of the prior subcube. Hence, the ending point of subcube \((0, 0, 0)\) should be \((00, 00, 01) = (00, 00, 10) - (0, 0, 1)\). Using the same method, we can determine the starting and ending points of the other subcubes and list them in Table 1.

Summarizing the above description, we will present a subcube connection algorithm for the n-dimensional Hilbert space-filling curve. Let \(G_i\) denote the \(i\)-th element of the \(n\)-bit Gray code sequence and \((\hat{b}_0x'_0, \cdots, \hat{b}_jx'_j, \cdots, \hat{b}_{n-1}x'_{n-1})\) be the index of point \(X\) in the orginal space. \(D\) is the subcube index \((\hat{b}_0, \cdots, \hat{b}_j, \cdots, \hat{b}_{n-1})\). \(S_i\) and \(E_i\) are the starting and ending points of the \(i\)-th subcube, separately. Function \(Hamming(S_i, j)\) inverts the \(j\)-th bit of \(S_i\). That is, \(S_i\) and \(Hamming(S_i, j)\) are of Hamming distance 1. Function \(Euclidean(X, k)\) returns index \((x_0, \cdots, x_k \pm 1, \cdots, x_{n-1})\) which is of Euclidean distance 1 from \(X = (x_0, \cdots, x_k, \cdots, x_{n-1})\). The output of the

### Table 1: Starting and ending points of \(H^2\) hypercubes (two categories)

| After Gray permutation | Category A | Category B |
|------------------------|------------|------------|
| subcube No. | subcube index | starting point | ending point | starting point | ending point |
| 0 | (0,0,0) | (0,0,0) | (0,0,1) | (0,0,0) | (0,0,1) |
| 1 | (0,0,1) | (0,0,0) | (0,1,0) | (0,0,0) | (0,1,0) |
| 2 | (0,1,1) | (0,0,0) | (0,1,0) | (0,0,0) | (0,1,0) |
| 3 | (0,0,0) | (0,1,1) | (1,1,1) | (1,0,1) | (0,0,0) |
| 4 | (1,1,0) | (0,1,1) | (1,1,1) | (1,0,1) | (0,1,0) |
| 5 | (1,1,0) | (1,1,0) | (1,0,0) | (0,0,0) | (1,0,0) |
| 6 | (1,0,1) | (1,1,0) | (1,0,0) | (1,1,0) | (1,0,0) |
| 7 | (1,0,0) | (1,0,1) | (1,0,0) | (1,0,1) | (1,0,0) |

\[\begin{align*}
[G^2 \otimes I_{2^{n-1}}](e^2_{n-1} \otimes e^2_{n-1} \otimes \cdots \otimes e^2_{n-1}) \\
= (e^2_{n-1} \otimes e^2_{n-1} \otimes \cdots \otimes e^2_{n-1} \otimes e^2_{n-1} \otimes \cdots \otimes e^2_{n-1}).
\end{align*}\]
algebra is the set of the starting and ending points of the subcubes.

00 Subcube Connection Algorithm:
01 $X = 0$;
02 for ($i = 0; i < 2^n; i++)$
03     $S_i = \| X; D = \| X$;
04     $j = 0$; found = false;
05     while ($j < n$ ∧ ¬found)
06         $X' = D \| E_i$;
07         $k = 0$;
08         while ($k < n$ ∧ ¬found)
09             $X'' = \text{Euclidean}(X', k)$;
10             if ($X'' == G_{s1}$)
11                 $S_i$ is the starting point of $i$-th subcube;
12                 $E_i$ is the ending point of $i$-th subcube;
13                 $X = X''$;
14                 found = true;
15             else $k = k + 1$;
16         $j = j + 1$;
17     }

The subcube connection algorithm can generate some hypercubes of different routes. For example, there are two $H^3_2$ of different starting and ending points listed in Table 1 and the figures in Fig. 1 (b), (c).

We select one of hypercubes and use to perform the coordinate transformation. This procedure includes two steps. Firstly, we operate the origin of the coordinates and the starting point of the subcube with XOR. The results of true status mapping coordinates which are applied reverse permutation to operate a binary complement. Secondly, we perform the starting and ending points coordinate with XOR. The result of true status mapping coordinate is applied stride permutation and the mapping coordinate rotated left to $x_0$ coordinate.

We take the the subcube (1, 0, 0) as the example. The starting and ending points of the subcube (1, 0, 0) are (1, 0, 1) and (1, 0, 0). At first, (0, 0, 0) XOR (1, 0, 1) is (1, 0, 1) and therefore $x'_0$ and $x'_2$ are separately applied reverse permutation. We use the tensor product formula $(J_{2^{n-1}} \otimes I_{2^n} \otimes J_{2^{n-1}})$ to explain the operation. The next step, (1, 0, 1) XOR (1, 0, 0) is (0, 0, 1) and using stride permutation to rotate $x'_2$ left to $x'_0$. The tensor product formula of the operation is $(L_{2^{n-1}} \otimes I_{2^n} \otimes L_{2^{n-1}})$.

Through the preceding steps, we obtain the coordinate transformation of the subcube (1, 0, 1) and the tensor product formula is $(L_{2^{n-1}} \otimes I_{2^n} \otimes L_{2^{n-1}})$. The tensor product formula is described as following:

$$(L_{2^{n-1}} \otimes I_{2^n} \otimes L_{2^{n-1}}) \otimes (J_{2^{n-1}} \otimes I_{2^n} \otimes J_{2^{n-1}})$$

$$(e_{X_{21}} \otimes e_{X_{21}} \otimes e_{X_{21}})$$

$$(L_{2^{n-1}} \otimes I_{2^n} \otimes L_{2^{n-1}})$$

$$(e_{X_{21}} \otimes e_{X_{21}} \otimes e_{X_{21}})$$

$$(e_{X_{21}} \otimes e_{X_{21}} \otimes e_{X_{21}})$$

$$(e_{X_{21}} \otimes e_{X_{21}} \otimes e_{X_{21}})$$

$$(e_{X_{21}} \otimes e_{X_{21}} \otimes e_{X_{21}})$$

The point $(x'_0, x'_1, x'_2)$ is changed to $(-x'_2, -x'_0, x'_1)$. The coordinate transformation of the other subcubes can be performed by the same procedure.

There are five types of subcube orientation in the coordinate transformation listing in Table 2 and their tensor product formulas are described as the followings.

1. Type I: $(L_{2^{n-1}} \otimes I_{2^n} \otimes L_{2^{n-1}})$ denotes $(x'_0, x'_1, x'_2) \Rightarrow (x'_2, x'_0, x'_1)$.
2. Type II: $(I_{2^n} \otimes L_{2^{n-1}} \otimes I_{2^n})$ denotes $(x'_0, x'_1, x'_2) \Rightarrow (x'_1, x'_2, x'_0)$.
3. Type III: $(I_{2^n} \otimes J_{2^{n-1}} \otimes J_{2^{n-1}})$ denotes $(x'_0, x'_1, x'_2) \Rightarrow (-x'_0, -x'_1, -x'_2)$.
4. Type IV: $(I_{2^n} \otimes L_{2^{n-1}} \otimes I_{2^n})$ denotes $(x'_0, x'_1, x'_2) \Rightarrow (-x'_1, x'_2, x'_0)$.
5. Type V: $(L_{2^{n-1}} \otimes I_{2^n} \otimes L_{2^{n-1}})$ denotes $(x'_0, x'_1, x'_2) \Rightarrow (-x'_0, x'_2, -x'_1)$.

We can use the $R^S_n$, a tensor product formula, $R^{S_1}_{r_1} \oplus R^{S_1}_{r_1} \oplus \cdots \oplus R^{S_1}_{r_1}$ to denote the coordinate transformation for the construction of $n$-dimensional Hilbert space-filling curves. Hence, the coordinate transformation for the 3-D Hilbert space-filling curve is defined as $R^S_n = R^{S_3}_{r_1} \oplus R^{S_3}_{r_1} \oplus R^{S_3}_{r_1} \oplus R^{S_3}_{r_1} \oplus R^{S_3}_{r_1} \oplus R^{S_3}_{r_1} \oplus R^{S_3}_{r_1} \oplus R^{S_3}_{r_1}$.

After applying the coordinate transformation, each of the subcubes are connected adjacent.
4.4 Recursive Construction

Finally, we recursively construct the $2^n \times H_{r-1}^n$ $n$-dimensional Hilbert space-filling curves corresponding to the $2^n$ subcubes. The recursive permutation is applied to the elements in each of the subcubes indexed in a linear order resulting from the previous steps.

We summarize the recursive tensor product formula of the $n$-dimensional Hilbert space-filling curve permutation as below:

$$H_{r}^n = G_{r}^n,$$

$$r > 1: \quad H_{r}^n = (I_{2^n} \otimes H_{r-1}^n) R_r^3 G_r^3 B_r^3$$

$$= (I_{2^n} \otimes H_{r-1}^n) (R_0^{31} \oplus R_1^{31} \oplus \cdots \oplus R_{r-1}^{31})$$

$$\left(G_r^2 \otimes I_{2^{n(r-1)}}\right) \left( \prod_{i=0}^{r-2} \left( I_{2^{n+i}} \otimes L_2^{2^{n+i-1}} \oplus I_{2^{n+i-1}} \right) \right).$$

When $n = 3$, the recursive tensor product formula of the 3-dimensional Hilbert space-filling curve is as below:

$$H_3^1 = G_1^3,$$

$$r > 1: \quad H_3^r = (I_{2^3} \otimes H_{r-1}^3) R_3^1 G_3^1 B_3^1$$

$$= (I_{2^3} \otimes H_{r-1}^3) \left( R_0^{31} \oplus R_1^{31} \oplus R_2^{31} \oplus R_3^{31} \right)$$

$$\left(G_3^2 \otimes I_{2^{n(r-1)}}\right) \left( I_{2^3} \otimes L_2^{2^{n-1}} \otimes I_{2^{n-1}} \right).$$

The recursive tensor product formula of the Hilbert space-filling curve permutation can be expanded repeatedly to derive the iterative tensor product formula as below formula. The formula can be proved using mathematical induction.

For $r \geq 1$,

$$H_{n+1}^r = \prod_{i=0}^{r-1} I_{2^n} \oplus \left( R_0^{31} \oplus R_1^{31} \oplus \cdots \oplus R_{r-1}^{31} \right)$$

$$\left(G_r^2 \otimes I_{2^{n(r-1)}}\right) \left( I_{2^n} \otimes L_2^{2^n} \otimes I_{2^{n-1}} \right).$$

Iterative tensor product formulas can be directly translated into high-level programming language programs. We will explain program generation in Sect. 5.

In the next subsection, we will describe the tensor product formula for the 4-D Hilbert space-filling curve and transform one point index of $2^2 \times 2^2 \times 2^2 \times 2^3$ elements to the Hilbert space-filling curve index.

4.5 Construction of 4-D Hilbert Space-Filling Curves

Using the curve construction algorithm to generate the starting and ending points of each subcubes, there are nine coordinate transformation types for the 4-D Hilbert space-filling curve and they are listing in Table 3.

According to the recursive tensor product formula of the $n$-dimensional Hilbert space-filling curve, we can derive the formula of the 4-D Hilbert space-filling curve. The formula is as below:

$$H_4^1 = G_4^1,$$

$$r > 1: \quad H_4^r = (I_{2^4} \otimes H_{r-1}^4) R_4^1 G_4^1 B_4^1$$

$$= (I_{2^4} \otimes H_{r-1}^4) \left( R_0^{31} \oplus R_1^{31} \oplus R_2^{31} \oplus R_3^{31} \right)$$

$$\left(G_4^2 \otimes I_{2^{n(r-1)}}\right) \left( I_{2^4} \otimes L_2^{2^{n-1}} \otimes I_{2^{n-1}} \right).$$
We take an example to explain the procedure for the construction method for the 4-D Hilbert space-filling curve. Suppose, the $2^2 \times 2^2 \times 2^2 \times 2^2$ data elements stored in row major computer memory, and the binary index of the 75-th element is (01, 00, 10, 11).

Applying block allocation on the element, we obtain the point locates in the subcube $(0, 0, 1, 1)$ and the index is $(1, 0, 0, 1)$. The next step is Gray permutation. The mapping of Gray permutation is $(0, 0, 1, 1) \Rightarrow (0, 0, 1, 0)$, and after operating Gray permutation the index of the subcube is changed to $(0, 0, 1, 0)$. From the curve construction algorithm, the starting and ending points of the subcube $(0, 0, 1, 0)$ is $(0, 0, 0, 0)$ and $(0, 0, 1, 0)$. We operate $(0, 0, 0, 0) \text{ XOR } (0, 0, 0, 0)$ and $(0, 0, 0, 0) \text{ XOR } (0, 0, 1, 0)$. The former operation derives the true status mapping coordinates of (1, 0, 1, 0) which are applied reverse permutation and the latter operation derives the true status mapping coordinate which is used the stride permutation to rotate left to $x_0$ coordinate. At this stage, the result obtained is (0, 1, 0, 1).

After recursively apply the procedure to construct this point, the index of the 75-th element (01, 00, 10, 11) is transformed to the 36-th element (00, 10, 01, 00) of the $2^2 \times 2^2 \times 2^2 \times 2^2$ Hilbert space-filling curve.

5. Program Generation

Both the recursive and iterative tensor product formulas can be used to generate programs for the $n$-dimensional Hilbert space-filling curves. In this paper, we present program generation of the iterative formula for the Hilbert space-filling curve. We use the syntax of C language to illustrate it can be any other data type.

The function interface of the iterative tensor product formula is given as $void$ hilbert3D(int $r$, int *a), where $n$ is the parameter denoting the problem size $2^r \times 2^r \times 2^r$ and $a$ is a pointer pointing to the starting address of the input array of $2^r \times 2^r \times 2^r$ elements. When the function returns, the result is stored in the array pointed by $a$. The iterative program is as follows:

```
01 void hilbert3D(int r, int int[] a) {
02 int i, n, li, jl, kl;
03 int b[pow(8, r)];
04 for (i = 0; i < r; i++) {
05 for (n = 0; n < pow(8, i); m++) {
06 for (li = 0; li < pow(2, r-i-1); li++) {
07 for (jl = 0; jl < pow(2, r-i-1); jl++) {
08 for (kl = 0; kl < pow(2, r-i-1); kl++) {
09 // Case 4: $(i_0, j_0, k_0) = (1, 0, 0)$,
10 // $(i_0', j_0', k_0') = (1, 1, 1)$
11 b[m+pow(8, r-i)+i1*pow(4, r-i)+j1*pow(2, r-i)+k1+pow(8, r-i-1)*4];
12 }
13 }
14 }
15 }
16 }
17 }
```

The for loop in Line 4 is the iteration corresponding to matrix product $\prod_{i=0}^{r-1}$ in the iterative formula. The tensor basis $e_1^r \otimes e_2^r \otimes e_3^r$ of the input data for the iterative formula $H^r_2$ is factorized to $e_1^r \otimes e_0^e_1^r \otimes e_1^e_2^r \otimes e_0^e_1^r \otimes e_1^e_3^r \otimes e_0^e_1^r \otimes e_2^r \otimes e_0^e_1^r \otimes e_3^r \otimes e_0^e_1^r$. The for loops in Lines 5 to 8 are corresponding to $e_0^e_1^r$, $e_1^e_2^r$, $e_2^r$, $e_3^r$, respectively. The bases $e_0^e_1^r$, $e_1^e_2^r$, and $e_3^r$ are implemented as eight cases $(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0), (1, 1, 1), (1, 1, 1)$, respectively.

In Line 10, the intermediate result in working array $b$ is copied to array $a$. From the 3-D Hilbert space-filling curves algorithms, the time and space complexities are $O(8^r)$ and $O(8^r)$, respectively, because it scans the entire $2^r \times 2^r \times 2^r$ data space and performs data movement.

6. Conclusions

In this paper, we develop a novel construction method for $n$-dimensional Hilbert space-filling curves and using tensor product to describe the construction method. The $n$-dimensional Hilbert space-filling curve construction method is expressed in both recursive and iterative tensor product formulas. Program generation of the iterative tensor product formula is also explained.

Space-filling curves are used in various applications such as R-tree indexing, image compression, and VLSI component layout. For an application such as image compression, data collection can be rearranged according to $n$-dimensional space-filling curve permutations to enhance locality relationship and improve compression ratio. For some other applications such as R-tree indexing, only data points in an R-tree are needed to compute their space-filling curve indices. The program in this paper may be used to generate a mapping table of the space-filling
curve permutation. However, it carries a burden of storage cost. We are going to use this concise, clear methodology to develop other n-dimensional space-filling curves, such as Moore curves, Peano’s curves, and the isomorphism of Peano’s curves, that can be applied to appropriate applications.

References

[1] S. Biswas, “Hilbert scan and image compression,” Proc. 15th International Conference on Pattern Recognition, vol.3, pp.207–210, 2000.
[2] A.R. Butz, “Space filling curves and mathematical programming,” Information and Control, vol.12, no.4, pp.314–330, 1968.
[3] A.R. Butz, “Convergence with Hilbert’s space filling curve,” J. Comput. Syst. Sci., vol.3, no.2, pp.128–146, 1969.
[4] M.-H. Fan, C.-H. Huang, Y.-C. Chung, J.-S. Liu, and J.-Z. Lee, “A programming methodology for designing parallel prefix algorithms,” Proc. 2001 International Conference on Parallel Processing, pp.463–470, 2001.
[5] D.M. Gavrila, “R-tree index optimization,” Proc. Sixth International Symposium on Spatial Data Handling, vol.2, pp.771–791, Edinburgh, Scotland, 1994.
[6] A. Graham, Kronecker Products and Matrix Calculus: With Applications, Ellis Horwood Limited, 1981.
[7] D. Hilbert, “Über die stetige abbildung einer linie auf Flächenstück,” Mathematische Annalen, vol.38, pp.459–460, 1891.
[8] C.-H. Huang, J.R. Johnson, and R.W. Johnson, “A tensor product formulation of Strassen’s matrix multiplication algorithm,” Applied Math Letters, vol.3, no.3, pp.104–108, 1990.
[9] H.V. Jagadish, “Linear clustering of objects with multiple attributes,” Proc. 1990 ACM SIGMOD International Conference on Management of Data, pp.332–342, 1990.
[10] J.R. Johnson, R.W. Johnson, D. Rodriguez, and R. Tolimieri, “A methodology for designing, modifying and implementing Fourier transform algorithms on various architectures,” Circuits Systems Signal Process, vol.9, no.4, pp.450–500, 1990.
[11] R.W. Johnson, C.-H. Huang, and J.R. Johnson, “Multilinear algebra and parallel programming,” J. Supercomputing, vol.5, no.2–3, pp.189–217, 1991.
[12] S. Kamata, R.O. Eason, and Y. Bandou, “A new algorithm for N-dimensional Hilbert scanning,” IEEE Trans. Image Process., vol.8, no.7, pp.964–973, 1999.
[13] S. Kamata, M. Niiim, R.O. Eason, and E. Kawaguchi, “An implementation of an N-dimensional Hilbert scanning algorithm,” Proc. 9th Scandinavian Conference on Image Analysis, pp.431–440, 1995.
[14] I. Kamel and C. Faloutsos, “On packing R-trees,” Proc. Second International Conference on Information and Knowledge Management, pp.490–499, 1993.
[15] I. Kamel and C. Faloutsos, “Hilbert R-tree: An improved R-tree using fractals,” Proc. Twentieth International Conference on Very Large Databases, pp.500–509, 1994.
[16] D. Kontos, V. Megaloioikonomou, N. Ghubade, and C. Faloutsos, “Detecting discriminative functional MRI activation patterns using space filling curves,” Proc. 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp.963–967, 2003.
[17] S.T. Leutenegger, J.M. Edgington, and M.A. Lopez, “STR: A simple and efficient algorithm for R-tree packing,” Proc. Thirteenth International Conference on Data Engineering, pp.497–506, 1997.
[18] S.T. Leutenegger and M.A. Lopez, “The effect of buffering on the performance of R-trees,” Knowledge and Data Engineering, vol.12, no.1, pp.33–44, 2000.
[19] C. Li, H. Zhu, and N. Wang, “Fast n-dimensional hilbert mapping algorithm,” International Conference on Computational Sciences and Its Applications, pp.507–513, 2008.
[20] S.-Y. Lin, C.-S. Chen, L. Liu, and C.-H. Huang, “Tensor product formulation for Hilbert space-filling curves,” J. Information Science and Engineering, vol.24, no.1, pp.261–275, 2008.
[21] C.-B. Liu, C.-H. Huang, and C.-L. Lei, “Design and implementation of long-digit Karatsuba’s multiplier using tensor product formulation,” Proc. Ninth Workshop on Compiler Techniques for High-Performance Computing, pp.23–30, 2003.
[22] X. Liu and G.F. Schrack, “An algorithm for encoding and decoding the 3-D Hilbert order,” IEEE Trans. Image Process., vol.6, no.9, pp.1333–1337, 1997.
[23] G. Melnikov and A.K. Katsaggelos, “A non-uniform segmentation optimal hybrid fractal/DCT image compression algorithm,” Proc. 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, vol.5, pp.2573–2576, 1998.
[24] N. Memon, D.L. Neuhoff, and S. Shen, “An analysis of some common scanning techniques for lossless image coding,” IEEE Trans. Image Process., vol.9, pp.1837–1848, 2000.
[25] B. Moon, H.V. Jagadish, C. Faloutsos, and J.H. Saltz, “Analysis of the clustering properties of the Hilbert space-filling curve,” IEEE Trans. Knowl. Data Eng., vol.13, no.1, pp.124–141, 2001.
[26] B. O’Sullivan, Applying partial evaluation to VLSI design rule checking, 1995.
[27] G. Peano, “Sur une courbe qui remplit toute aire plane,” Mathematische Annalen, vol.36, pp.157–160, 1890.
[28] J. Quinqueton and M. Berthod, “A locally adaptive Peano scanning algorithm,” IEEE Trans. Pattern Anal. Mach. Intell., vol.3, no.4, pp.403–412, 1981.
[29] J. Romeu and S. Blanch, “A three dimensional hilbert antenna,” Proc. IEEE Antennas and Propagation Society International Symposium, vol.4, pp.550–553, 2002.
[30] S. Rovetta and R. Zunino, “VLSI circuits with fractal layout for spatial image decorrelation,” Proc. 1999 IEEE International Symposium on Circuits and Systems, vol.4, pp.110–113, 1999.
[31] H. Sagan, Space-Filling Curves, Springer-Verlag, 1994.
Min-Hsuan Fan received his B.S. degree in Mathematics from National Tsing Hua University and M.S. degree in Computer Science from University of Maryland, Baltimore County, in 1985 and 1990, respectively. His research interests include parallel processing and programming methodology.

Chua-Huang Huang received the B.S. degree in mathematics from Fu-Jen University, Taipei, Taiwan, in 1974, the M.S. degree in computer science from University of Oregon, Eugene, Oregon, in 1979, and the Ph.D. degree in computer science from the University of Texas at Austin, Austin, Texas, in 1987. From 1979 to 1982, Dr. Huang worked as an assistant researcher in the Telecommunication Laboratories, Ministry of Communication, R.O.C. He was the project manager of a Chinese terminal project. Dr. Huang was an assistant professor in the Department of Computer and Information Science, the Ohio State University, from 1987 to 1993 and an associate professor from 1993 to 1997. From 1997 to 2000, Dr. Huang was a professor at the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. Since 2000, Dr. Huang has been a professor at the Department of Information and Engineering, Feng Chia University, Taichung, Taiwan. Dr. Huang’s research interests include algorithms, software engineering, and RFID.