A Sector-Labeling for generating the Hilbert Space-filling Curve and Its Intention

Santosa Slamet[1] and Tohru Naoi[2]
[1]Department of Electronics and Applied Computer Science
Gifu University, Gifu, Japan
Tel. +81-58-230-1111, Fax.: +81-58-230-1895, E-mail: santosa@nao.info.gifu-u.ac.jp
[2]Department of Electronics and Applied Computer Science
Gifu University, Gifu, Japan
Tel. +81-58-230-1111, Fax.: +81-58-230-1895, E-mail: naoi@nao.info.gifu-u.ac.jp

Abstract: Many scientific applications include manipulation of data points lying in a space. We describe a method, based on sector labeling to generate a space-filling curve for partitioning such given data points. Our method is straightforward and flexible, equipping a one-one correspondence between point-values on the curve and data points in space in more efficient than designated methods found in the literature. It is widely believed that the Hilbert curve achieves the desired properties on linear mappings due to the locality between data points. Therefore we focus on the Hilbert curve since, later on, we identify it as the most suitable for our application. We demonstrate on using our method for the data particles of an n-body simulation that based on Barnes-Hut algorithm.

1. Introduction

The problem of assigning a large of data points to massively parallel computing systems is one of important tasks in the field of parallel processing. Moreover, in the dynamic of non-uniform data points that arising in many scientific applications, including particle simulations [1,2] and computational mechanics [3]. In such applications, computation involves nearby points through aggregations of data points. Therefore the adjacent points should be assigned into the same processor in order to minimize communications while computation. To solve such a load assignment efficiently, the large data points have to be divided in balance while at the same time locality of data points in each partition should be maintained. Wei Ou and Sanjay Ranka [4] and Hanan Samet [5] have shown that coordinate bit interleaving constructs a space-filling curve that provides good quality partition for an irregular problem. Warren and Salmon [6] have proposed a similar technique for partitioning non-uniform data points (particles) in an N-Body simulation. The technique is basically used a mapping by indexing so that the data points continue on a line and lying close each other. A key index is used to distinguish a point, constructed by interleaving it coordinates and the sorted keys produce a particular sequence in Z-curve. However, since it jumps conduct the discontinuity of spaces, partitions may not preserve enough locality information of the data points. Pilkington et al [7] have been suggested to use a Hilbert curve through attempting on mapping a collection of points that lie on a uniform grid. Each point is assigned an index based on 'inverse' Gray-code mapping converted from its coordinates, which represents its approximate position on the curve. However, to achieve such distinct indices for all the given data points many sector rotations are needed, which may reduce the degree of it possible implementation.

We propose in this paper a new alternative technique, based on sector labeling to generate a Hilbert curve, for partitioning the given data points. This approach leads to simplicity, involve no complex bit interleaving operations. It is also quite flexible, it handles regular and irregular data points in the same way. Given irregular data points lying in a rectangular domain, we need to find a decomposition for which each point is then located in a different sector. We label the decomposed sectors in such a way that the ordering of points representing one-dimensional curve is established by their Hilbert values. The curve imposes an order on points located in the divided sectors i.e., a point assumes the Hilbert value of its smallest enclosing sector. Having the ordered data points, partitioning is then can be done as was described in [6] and [7]. Note that, our Hilbert curve representation is organized in a tree structure. In addition, we describe a technique for arranging the data points across partitions. Each partition can has evenly balanced data quantity but the length of a curve section shall vary according to the local density of data points. We had examined that our Hilbert curve based partitioning provides a useful technique, which facilitates on easy linking the nodes of tree.

This paper is organized as follows. Section 2 describes a brief space-filling curve in conjunction with partitioning. Section 3 presents in detail the domain decomposition and how the mapping is computed. Section 4 shows a practical application on using the Hilbert curve. We made a brief locality comparison with that occupy Z-curve ordering in section 5. Section 6 includes our conclusions.

2. Space-filling curves and Partitioning

Methods for manipulating space-filling curves are typically based on a geometric manipulation of how the domain decomposes into multiple smaller versions and linked together. For ease of illustration, we deal with the curve's generation as labeling the divided sectors in two dimensions, since it expresses the concept of space-filling curves in a simple manner.

Space-filling curve can also be thought of how to order the decomposed sectors in such a way that labeling each sector appropriately perform the self-similar construction of the curve. For a rectangular domain decomposition of level k, one can associate a one-to-one mapping $L_k : \{1, \ldots, 2^k\} \times \{1, \ldots, 2^k\} \rightarrow \{1, \ldots, 2^{2k}\}$. This mapping specified labeling of level k that linearly orders 2^{2k} sectors and defines a curve on the original rectangular domain. For an irregular data points lying in such a rectangular domain we can therefore continuously subdivide the domain till all divided sectors contain no more than one point. Thus the data points are ordered and there is a one-one correspondence between labels on the curve and the points in space domain. There are many space-filling curves in the literature [8] but we concentrate on a Hilbert space-filling curve for our partitioning, since it is widely believed that the Hilbert curve achieves the desired properties on linear mappings [9].

Sector labeling can be performed on the decomposed sectors such that the sequence of labeled sectors $s_1, s_2, \ldots, s_n$ indicates their positions on the Hilbert curve. As a result of the recursive fashion in which a rectangular domain is decomposed, such a mapping can be expressed as a tree structure. In practical applications of irregular data points the decomposition is terminated after k levels such that the smallest enclosing sectors, with $n \leqslant 2^{2k}$, construct the curve of order k. Since each sector contains a point, the order in which the curve visits the smallest enclosing sectors also determines the order in which the curve visits the points.

Partitioning the data points includes the generation of a Hilbert curve, i.e., ordered the decomposed sectors by a sector labeling and cutting the curve into partitions in such a way that balance the loads upon a particular application. Having such partitions, the data points are then assigned into processors, while the locality is preserved. This leads into possibility on reducing communication overhead since the Hilbert curve does not involve spatial discontinuities.

3. Sector decomposition and its labeling

In general, Hilbert curve starts with a basic path that can be drawn as a U-shape on a two-dimensional domain. The sector domain enclosing the given data points is subdivided into four sub-sectors, to which the basic path visits exactly once without crossing itselves. The basic path is called as order 1. To derive a curve of order k, each vertex of the basic curve is replaced by the curve of order $k-1$, which may appropriately rotated and/or reflected. Figure 1 shows examples of two-dimensional Hilbert curve. We clarify that a curve is in order k when the curve associates with the decomposition of level k.

We refer to the labels as the Hilbert values, which are used to order the divided sectors. Such an ordering is continuous, due to continuity of the Hilbert curve, i.e. sectors adjacent in the order are closest in space. Since a single point assumes a Hilbert value of its smallest enclosing sector, the sector ordering induces the ordering

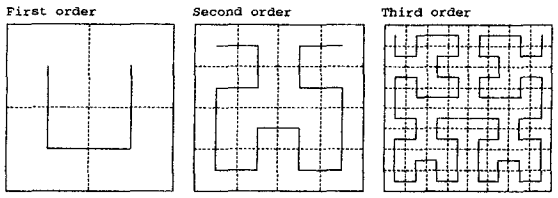

Figure 1: Two dimensional Hilbert curve

of data points in the sequence of the curve. We used the following strategy to determine subdivision level to ensure that each point is located in a different sector. Suppose a divided sector at any level k that has two nearest points close to the opposite ends of a diagonal. Clearly, the smallest sector that encloses each point of these two points is of level $k+1$, at which the recursion is then terminate. This could be done by first finding nearest distance between two points of the given data points and comparing the distance with the diagonal of being built sector.

The decomposition starts with defining an original sector, a square $[0,1]^2$ that large enough to contain the given data points, which then repeatedly subdivides into four sub-sectors. Hence, each sector has four equal size child-sectors, and these divisions build a quad-tree with the root corresponds to the original sector. A node of the quad-tree includes information concerning the corresponding sector i.e., the anchor x, y, the node's lengths lx, ly, the number of data points n, a label in characters, a pointer to the given data points $*points$ and four pointers to the child-sectors $*subsector[4]$. The following describes the sector decomposition, tree construction as well as labeling the divided sectors.

Input:
1. A set of data points (X, Y), stored in an array a
2. Domain sector contains the data points as root node

Output:
A list L of smallest enclosing sectors, organized as a queue

int **decompSector:**
1. Initialize a pointer $root = NULL$;
2. Create an empty list L;
3. $root$=**qBuild**$(\&L, x, y, lx, ly, n, a[], qfactor = 0.0)$;
4. Return 0;

node ***qBuild:** variable i: integer;
1. Declare a node's pointer p;
2. If (non empty L)
   dequeue (p, L); Endif;
3. If $(qfactor = 0.0)$
   assaign p to point to current node; {root node}
   $qfactor =$ **addPoint** $(p, a[], n)$;
   {plot points and find nearest distance}
   if (there are no valid point in root)
      free (p);
      return $NULL$;

```
    endif;
  Else
    if (current node's diagonal < qfactor)
        return NULL;
    endif;
    p = malloc (sizeof(node));
    addPoint (p, a[], n); {use current qfactor}
  Endif;
4. Label (p);
5. queue (L, p);
6. For (i = 0; i < 4; i++)
    p → subsector[i] =
    qBuild (&L, x, y, lx/2, ly/2, n, a[], qfactor);
  Endfor;
7. Return p;
```

```
double addPoint:
variable i,i: integer; factor = double;
1. Declare a node's pointer p;
2. For (i = 1; i < n; i++)
    Determine the data point in current node;
    realloc (p → points, (p → npoint+1) *
    sizeof(point));
    p → points[(p → npoint)++] = a[j];
  Endfor;
3. If (p → npoint > 1)
    Find distance between pair of data points;
    factor = minimum distance of data points;
  Endif;
4. Return factor;
```

Each time the function **qBuild** is called, the node is sub-divided to a greater level of details in which if v is a node at level k then v_1, v_2, v_3, v_4 are the divided sub-nodes, while the level growing from k to $k+1$. Our labeling procedure therefore can be described in the term of square matrix operation. Initially let root node be labeled 0. Let $M = [r][c]$ be a $2^n \times 2^n$ square matrix which each of it elements is a square matrix of $2^{n-1} \times 2^{n-1}$. At the first division, let $n = 1$ and the elements of M : $r_1c_1 = 1, r_1c_2 = 2, r_2c_1 = 0, r_2c_2 = 3$ are appended to the root node label that corresponding to the sector labeling of level $k = 1$. The sector labeling grows for greater levels in the following way. Define a square matrix

$$L = \begin{pmatrix} M & M \\ f(M) & g(M) \end{pmatrix}$$

The function $f(M)$ replaces $M = [r][c]$ by $M = [m+1-c][n+1-r]$ and the function $g(M)$ replaces $M = [r][c]$ by $M = [c][r]$ respectively. The labels of sectors at level $k+1$ are obtained by appending the elements of L to the labels of sectors at level k.

4. Application of the Hilbert-curve

We now describe how the Hilbert curve based partitioning in a practical application. We refer to the data points as the data particles in which their interactions treated based on Barnes and Hut (BH) algorithm [1]. Note that in BH algorithm, the method has two steps: upward evaluation of the center of mass for every node and downward evaluation for force calculation on each particle. The center of mass is computed recursively by calculating the sub-nodes center of mass first, and then uses the results to calculate the center of mass of a node. In the following, we assume that the data particles have been partitioned among processors of a parallel computer and we will limit the discussion on evaluation of the center of mass.

The sequential recursive fashion is outlined below. If a node v has sub-nodes v_i, calculate these sub-nodes center of mass, and then use the results to calculate the center of mass of v.

```
calculateCenterOfMass (node v):
    if v is a leaf then return;
    for each node v_i which is a sub-node of v
        calculateCenterOfMass (v_i);
        SUM weighted center of mass of v_i;
    SET center of mass of v;
    return;
```

Having in part the center of mass of nodes level $k-1$ computed in a processor, to compute its lower level the procedure introduces a request for center of mass of internal nodes computed in the other processors. Upon receipt such a broadcast message, a processor merges the data appropriately into local tree. The Hilbert curve adapts properly on updating the nodes, i.e. insertion of received data center of mass into local tree structure.

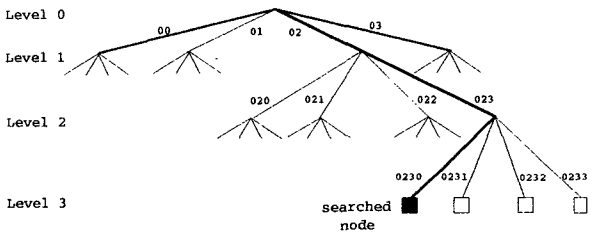

Figure 2: Searching on tree representing Hilbert curve

Each section is indicated by its lowest and highest Hilbert values, referred as *section-keys*, to which the curve will be concatenated. We describe insertion of nodes using an example, in terms of descending the tree representation of the Hilbert curve. Figure 2 shows a tree structure that the labeled nodes at each level represent the Hilbert curve. We assume that each processor has been identified its own section-keys, and a *right-insert* uses the highest value as the current-key, which in this example is the value is '0223'. Similarly, a *left-insert* can be done in the same manner. Note that the insertion precedes by searching in binary fashion for the next sub-nodes in tree labeled with the values which are greater than current-key. The search begins at the root, descends and terminates at the next-match, which in this example is the value '0230', as follow:

Tree level 0: The root node contains both the current-key and the next-match.

Tree level 1: A binary tree search for *right-insert* discards the nodes labeled '00' and '01', because the top two digits of the current-key are '02'. We can deduce that the next-match is might found in a node, which is a descendent of node labeled '02', since the value of final two digits of current-key is least than '33'. Therefore the search can proceed to the sub-nodes at level 2 pointed to by the node labeled '02'.

Tree level 2: The search now downed one level into the nodes at level 2. Due to the top three digits of the current-key that are '022', the search discards the nodes labeled '020' and '021'. Therefore the next-match presumably within a node that is the descendent of node labeled '022' or '023'. However, since the value of final digit of current-key is '3' and the nodes are logically ordered upon the sector decomposition, then the search therefore can reject the node labeled '022'.

Tree level 3: The search continues one level to the nodes at level 3 pointed to by the node labeled '023'. At this level the search found the next-match, the node labeled '0230'.

5. Locality preservation

We briefly compare locality behavior induced by two different partitioning strategies, which might fulfills the needs of parallel N-body simulation.

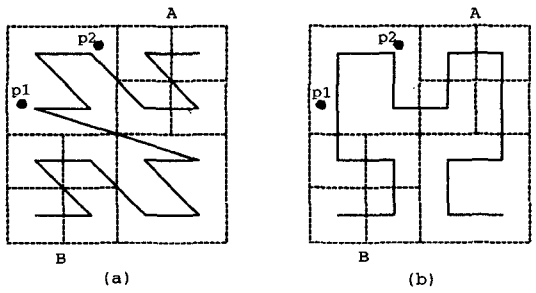

Figure 3: Partitioning based on space-filling curves

A discrete space-filling curve of level k, C_k : $\{1,\ldots,2^k\} \times \{1,\ldots,2^k\} \rightarrow \{1,\ldots,2^{2k}\}$ fulfills $d_E = (H(i), H(i+1)) = 1$, i.e. a segment $||(i+1) - i||$ represents an Euclidean distance on the curve in one unity. We mean by locality of a partition as if according to the ordering it holds that $||i - j||$ small then $d_E(i,j) = \sqrt{(x(i) - x(j))^2 + (y(i) - y(j))^2}$ shall also be small. Figure 3 shows partitioning schemes based on space-filling curves. We assume that the data particles have been partitioned among processors based on both Z-curve and the Hilbert curve respectively. In case of Z-curve (figure 3a), due to the distance on the curve, the data particles lying in sector B might partitioned into different processors with particle p_1 while those lying in sector A might partitioned into the same processor with particle p_2. This causes when traversing the node B for force calculation on p_1 in BH algorithm the data points are not found and need an exchange for those essential data points, while when calculating force on p_2 no communication is needed. This implies for the N-Body simulation, that a lot of communications are needed when computing the forces on particles about p_1. In contrast in the case of the distance on Hilbert curve (figure 3b), both the data particles lying in sector B and A are could be partitioned into the same processor with particle p_1 and p_2 respectively.

6. Conclusions

We have described a sector labeling for manipulating the Hilbert space-filling curve. The method leads to a simplicity and robust approach, manages to avoid complex bit interleaving operations, with certain flexibility in handles regular and irregular data points.

This approach leads to creating algorithms, e.g. finding nodes or points in the neigbor partitions, with respect to a space-filling Hilbert curve. Note that backtracking may take place but not always necessary and a next-match is guaranteed to exist. This is useful for particular applications, as we have shown an example in an N-Body simulation.

References

[1] J. Barnes and P. Hut, "A Hierarchical O(N log N) Force-calculation Algorithm," *Nature*, Vol. 324, pp. 446-449, 1986.

[2] L. Greengard and V. Rokhlin, "A Fast Algorithm for Particle Simulations," *J. Comp. Physics*, Vol. 73, pp. 325-348, 1987.

[3] B. Hendrickson and K. Devine, "Dynamic load balancing in computational mechanics," *Comp. methods in applied mechanics and engineering*, Vol. 184, pp. 485-500, 2000.

[4] CW. Ou and S. Ranka, "Parallel Remapping Algorithm for Adaptive Problems," *IEEE Frontier 95, The fifth Symposium on The Massively Parallel Computations*, pp. 367-374, 1995.

[5] H. Samet, "The Design and Analysis of Spatial Data Structure," *Addison Wesly Publishing Company, Inc.*, 1990.

[6] MS. Warren and JK. Salmon, "A Parallel Hashed Oct-Tree N-Body Algorithm," *Proceeding Supercomputing, The best student paper*, 1993.

[7] JR. Pilkington and SB. Baden, "Dynamic Partitioning of Non-uniform Structured Work-load with Space-filling Curves," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7(3), pp. 288-299, 1996.

[8] B. Moon, HV. Jagadish, C. Faloutsos and JH. Saltz, "Analysis of tha Clustering Properties of Hilbert Space-filling Curve," *IEEE Transaction on Knowledge and Data Engineering*, 1996.

[9] DJ. Abel and DM. Mark, "A Comparative Analysis of Some Two-dimensional Orderings," *Int. J. Geographical Information Systems*, Vol. 4(1), pp. 21-31, 1990.