Sector-Labeling for the Hilbert Curve Construction and Its Application to Mapped Data Retrieval Algorithms

Santosa Slamet, Tadahiro Matsumoto, Tohru Naoi and Munehiro Goto
Department of Electronics and Applied Computer Science
Gifu University, Japan.
Tel.: +81-58-230-1111, Fax.: +81-58-230-1895

Abstract

Parallel computation for data points plays a significant role in many scientific applications like many-body simulations. In such a simulation, the partitioning of the given data points, such that the locality of the data points is preserved is needed. In the computation, the data points are partitioned based on a kind of labeling or ordering. It is widely believed that the Hilbert curve on the data space is most suitable for the ordering of the data points. This paper describes a sector labeling method of the given data points. This method generates a space-filling curve, which gives a label on each data point such that the data points are mapped in a linearized order that preserves the locality of the data points. The label has the property similar to that of Hilbert curve and is very suitable for the computation of the data points. This paper verifies that the data retrieval algorithms and the labeling method work very well in our parallel computation of data points.

Keywords: Spatial information, Partitioning the data points, Locality preservation, Parallel computation of data points.

1. Introduction

Efficient manipulations of the data points seem to be particularly dominant in many important scientific applications. Such examples include computational molecular dynamics [1], computational mechanics [2] and many-body simulations [3, 4, 5]. When running these simulation programs on a parallel computer, the data points must be partitioned and assigned into processors in such a way that each processor maintains the tasks in balanced load. Moreover, since the computations of such scientific applications usually involve the aggregation operation of nearby data points, a linearization that keeps locality is required.

Several partitioning algorithms for data points have been proposed that exploit the idea of recursively dividing the data points. The simplest is Orthogonal Recursive Bisection (ORB) proposed by Berger and Bukhari [6].

Recently, a more sophisticated partitioning technique that involves an ordering based on space-filling curves have been intensively studied as an important field in scientific computation and many algorithms have been developed. Wei Ou and Sanjay Ranka [7] and Samet [8] have shown that the coordinate bit interleaving constructs a space-filling curve, called Z-curve that provides the partition of good quality for even irregular problems. Warren and Salmon [9] have proposed a similar technique for partitioning non-uniform data points (particles) in a many-body simulation. The technique basically uses an indexing in which a key is used to distinguish a point, constructed by interleaving integer coordinates of the given data points. For instance, a key $Z_p = x_1y_1z_1x_2y_2z_2x_3y_3z_3...x_ky_kz_k$ represents a data point $p =< x_1x_2x_3...x_k, y_1y_2y_3...y_k, z_1z_2z_3...z_k >$ on a Z-curve of order k. However, since Z-curve's jumps conduct the discontinuity of spaces, partitions may not preserve enough locality information of the data points. Pilkington et al. [10] have suggested to use a Hilbert curve for partitioning a collection of data points that lie on a uniform grid. In their method, the domain where all data points lie is decomposed into smaller sectors in which each sector has at most one data point. For a two-dimensional domain, for example, a square domain will be divided into four sectors of equal size, i.e. each sector is

divided in sequel until the sub-sector contains at most one data point. In Pilkington's method, to assign a proper index to each data point, some sectors have to be rotated. Since these sector rotations are executed for every level of decomposition and the indexing of the data points requires complicated bookkeeping and enormous amounts of memory usage, the algorithm is difficult to implement.

We propose in this paper a new alternative technique, based on a Hilbert curve. This approach avoids complex bit interleaving operations as well as wasteful sector rotations. It is also quite flexible and it handles regular and irregular data points in the same way. Given irregular data points laying in a rectangular domain, we need to find a decomposition for which each point is located in a different sector. While the decomposition proceeds, we directly label the divided sectors in such a way that the label of sectors establishes the ordering of data points representing a one-dimensional Hilbert curve.

In parallel computation, the given data points ordered by the Hilbert curve are divided into several parts so that the workload of each partition is evenly balanced. Each partition can have a different number of data points according to local density of the data points. In a many-body simulation, the load-balance decision depends on how much workload is needed to compute the interaction between data points on each partition [11], however this problem is beyond the scope of this paper.

In the field of parallel many-body simulation, it is necessary to compute the force interactions between data points in such a way that we first have to find the data points which have strong interaction for the particular point. We describe a retrieval technique for data points mapped to the Hilbert curve and those meet such strong interaction criteria, which gives an important contribution for the computation described previously. However, our approach is general, hence it can be also applied to other fields of simulations.

2. Preliminaries

A space-filling curve is actually family of a curve that can be defined recursively. The ordered paths of a space-filling curve have similar shapes at different levels, i.e. a higher-level curve is constructed by adding segments of similar shapes to the previous curve in order to fill more the space. One example is Hilbert curve, which is the first geometrical representation of a space-filling curve [12]. For the shake of simplicity, in this paper we describe the method how to generate the Hilbert curve by labeling the divided sectors only in two dimensions. The related algorithm is described in detail in section 3.

The space domain is defined as a Cartesian product space that is large enough to contain the given data points. This domain is then recursively subdivided into four sectors of equal size until each sector contains at most one data point. We called such the divided sectors as the smallest enclosing sectors. A labeling of the divided sectors is then completed in such a way that after k levels of decomposition, a labeling of the divided sectors specifies a mapping $L_k : \{1,...,2^k\} \times \{1,...,2^k\} \rightarrow \{1,...,2^{2k}\}$ that linearly orders 2^{2k} sectors and defines the Hilbert curve on the original domain. The Hilbert curve is regarded as the ordering of the smallest enclosing sectors with the level of subdivision k designates the order of the curve. The curve imposes an order on data points located in the divided sectors i.e., a point assumes a label value of its smallest enclosing sector.

Because the curve is generated recursively, the Hilbert curve can be expressed as a tree structure. In our implementation of the algorithm, the structure of a sector is declared as a tree node that has four pointers and we note that the terms sector and node are interchangeable. We begin the tree construction by initializing the space domain as the root of the tree. As each sector is divided into four sub-sectors, each of pointers can be initiated to point to a sub-sector and a sub-tree is then built. This process can then be continued for every sector until every sector has at most one data point. Therefore, the height of the corresponding tree is finite and equal to the order of the curve. The leaves of the tree corresponding the Hilbert curve represent the smallest enclosing sectors and so the tree greatly aids the development of algorithms, which are used to facilitate the execution of the data retrieval.

The data retrieval depending on some kind of locality is necessary in parallel computation such as a many-body problem. The locality is defined by the geographical distance between data points or other metrics. In a many-body

problem, Lars Hernquist has defined formally such a locality by using a simplified geometry in [5], from which we can draw a region. In the parallel computation, we need to calculate the interaction on a data point from the other data points. We refer the region that contains such data points as a question range, in which we need to seek and retrieve such data points to proceed the computation.

In a two-dimensional case, the boundaries can be expressed by the region of x and y coordinates $< L_1, L_2 >$ and $< U_1, U_2 >$. Then the question range is the square defined by the four points $[L_1,U_1]$, $[L_1,U_2]$, $[L_2,U_1]$and $[L_2,U_2]$. The data points within the square satisfy some given question and so are retrieved in the computation. Our labeling method to the Hilbert curve works just fine to retrieve the data points in the question range. We describe in more detail the data retrieval algorithms as well as the example of its execution in section 4.

3. Sector decomposition and its labeling

In general, the Hilbert curve starts with a basic path that can be drawn as a U-shape. The space domain is subdivided into four equal sectors, to which the basic path visits exactly once without crossing itself. The curve of order k is obtained by the composition of four instances of the curve of order $k-1$ and appropriate rotation and or reflection of U-shape. Such a curve can be constructed by giving a label to each subdivided sector such that the label sequence along the curve represents a Hilbert curve. We say that a curve is of order k when the curve associates with the decomposition of level k. Figure 1 shows examples of two-dimensional Hilbert curve. The 2^6 sectors are ordered by the Hilbert curve of order 3.

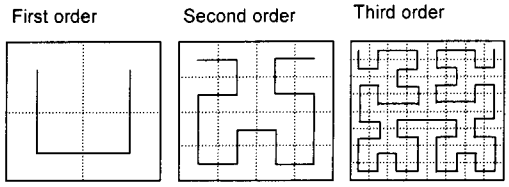

Figure 1: Two dimensional Hilbert curve

We use integers to represent the labels, which are used to order the divided sectors. Such an ordering preserves locality in two-dimensional space, because two neighbor's points of a point on the curve are always chosen from its four neighbors. Since the domain decomposition continues until the stage where each sector has at most one data point, the sector ordering induces the ordering of data points in the sequence of the Hilbert curve.

In what follows, the domain decomposition and the labeling method of the divided sectors for two-dimensional case are described. In this method, the decomposition starts with defining an original sector, the space domain of a square $[0,1]^2$, which is large enough to contain the given a set of n data points, and then repeatedly subdivides the sectors into four sub-sectors. Hence, each sector has four equal sized child-sectors, and these divisions build a quad-tree with the root corresponds to the original sector. In the following algorithm, a node of the quad-tree contains the information concerning the corresponding sector i.e., the lower-left sector x,y, the sector's lengths lx,ly, the number of data points n, a label in integers, a pointer to the given data points **points* and four pointers to the child-sectors **sub_sector[4]*.

Input:
1. A set of n data points, denoted by X,Y stored in an array a
2. The domain sector contains the given 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* = **quadBuild** *(&L, x, y, lx, ly, n, a[], qfactor = 0.0)*
4. Return 0;

node ***quadBuild:**
variable *i*: integer;

1. Declare a node's pointers p ;
2. **If** (non empty L)
   dequeue (p , L);
   Endif;
3. **If** (*qfactor = 0.0*)
   assign p to point to current node;{root node}

```
    qfactor = addPoint (p, a[], n);
    {plot data points and find nearest distance of
    points}
     if (there are no valid points in root)
        free (p);
        return NULL ;
     endif;
   Else
     if (current node's diagonal less than qfactor)
       retun 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 -> sub_sector[i] = quadBuild (&L, x, y, lx/2,
      ly/2, n, a[], qfactor);
   Endfor;
7. Return p;
```

```
double addPoint:
variable i, j: integer; factor: double;

1. Declare a node's pointer p;
2. For (i = 1; i < n; i++)
     determine which data point laying 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 X and Y between pair of data
   points;
     factor = minimum distance of the data points;
   Endif;
   Return factor;
```

In the above algorithm, whenever the function **quadBuild** 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, the child nodes, while the level growing from k to $k+1$. Our labeling procedure is directly made on the divided sectors by appending appropriate numbers to the parent sector's labels to get the child sector's labels. In contrast to that of [9], when the domain is divided into four sub-sectors, to give a distinctive index to a point, the divided sectors must be rotated in such a way that the values of sector coordinates as well as the coordinates of the data points must be kept to be restored later. Obviously, so many rotations need very complicated bookkeeping.

In the following we describe an additional explanation for our sector labeling method. In two dimensions, while a node is subdivided into four sub-nodes, we label these sub-nodes by appending the elements of a square matrix A to the label of their parent node. In general, the labels of nodes or sectors at level $k+1$ are obtained by appending the elements of A to the labels of sectors at level k. Initially, the root node is labeled as 0. Let A be a square matrix of $m \times m$, which initially at $m = 2$, the elements of A are set as $a[1][1] = 1$, $a[1][2] = 2$, $a[2][1] = 0$ and $a[2][1] = 3$. Appending the elements of A into the root label appropriately yields the labels of it sub-nodes, which are 00, 01, 02 and 03 respectively.

When the decomposition level grows from level k to $k+1$, the element of the label matrix A grows from $2^k \times 2^k$ to $2^{k+1} \times 2^{k+1}$. This is described in the following matrix substitution:

$$A \Leftarrow \begin{bmatrix} A & A \\ f(A) & g(A) \end{bmatrix}.$$

We used this matrix in our labeling procedure. The function $f(A)$ replaces the elements of $A[i][j]$ by $A[m+1-j][m+1-i]$ and the function $g(A)$ replaces the elements of $A[i][j]$ by $A[j][i]$ respectively. Figure 2 shows examples of such two-dimensional sector labeling, where "+" means to append the left term to each element of A from left side.

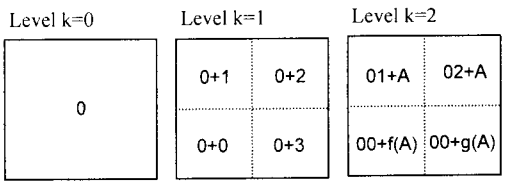

Figure 2: Two dimensional sector labeling

The extension of the sector labeling to the three-dimensional case is straightforward. The related sector is a cube in which the decomposition subdivides each cube into eight sub-cubes. The cube labeling therefore grows into $1..2^{3k}$ that can be implemented using a three-dimensional array.

4. Retrieval algorithms on the Hilbert curve

When defined a question range in the space domain according to some question associated with the locality like in [5], the Hilbert curve will probably enter to or leave the question range many times. The parts of the Hilbert curve within the question range are the curve sections corresponding to the partitions. These partitions may contain many data points, which have strong interaction to a focused data point in the meaning of the question range. Therefore, we have to find the partitions and retrieve the data points that lie to carry on the force computation on that data point.

In particular, we concern ourselves with the algorithms to identify which partitions may contain the matches, i.e. the partitions intersect with the question range from which the data points will be retrieved. Our strategy is to retrieve the data points by copying the data points satisfying the given question step by step according to the order of points on the Hilbert curve. The partitions within the question range define a sub-set of data points. The retrieval process reduces this sub-set of data points and when the sub-set is empty the retrieval process is completed.

We control the retrieval process with a parameter *next-match* to which the lowest value or label is assigned. After removal of some data points a new lowest value of those remaining in the sub-set is computed. Calculation of the *next-match* is performed by a function **findNextMatch**, which is also used to determine the lowest value to the question range. In our implementation, we used variables *partition-key* and *exa-partition* to store the lowest value or label of point in the partitions and a partition in which local computation being considered respectively. We also used some other variables, which have the names meaningful with the descriptions. The data retrieval algorithms are shown as follows.

Algorithm 1: Finding a *next-match* of a question range

1 *current-search-sector* ⇐ the whole domain
2 *current-question-range* ⇐ the whole of the question range
3 *next-match* ⇐ 0
4 *partition-key* ⇐ the lowest value or label of a partition
5 *current-tree-level* ⇐ initially the root level, i.e. level 0
6 **while** *current-tree-level* is lower or equal leaf level **do**
  determine in which sector of the *current-search-sector* the coordinates of *current-question-range* lie
7 $V \Leftarrow m$ digits taken from *partition-key* corresponding to the tree level
8 **call** a binary search of algorithm 2, to find the *current-sector* which intersects with the *current-question-range*, whose the label is minimum but higher or equal to V
9 **if** binary search failed to identify the *current-sector* **then**
  return FALSE
  endif
10 **if** *current-tree-level* ≠ the leaf level **then**
  current-question-range ⇐ *current-question-range* ∩ *current-sector*
  current-search-sector ⇐ *current-sector*
  endif
11 **if** V = the value of the *current-sector* **and** *current-search-sector* = *current-sector* **then**
  next-match ⇐ *partition-key*
  return TRUE
  {The partition-key lies within specified question range, therefore the *next-match* is identified}
  endif
12 *next-match* ⇐ *next-match* + ((the value of current-sector) << (m (*current-tree-level*))
  {Append the value of *current-sector* to the *next-match*}
13 *current-search-sector* ⇐ *current-sector*
14 *current-tree-level* ⇐ *current-tree-level* + 1
15 **if** $V <$ the value of the *current-sector* **then**
  BREAK out of the loop and go to loop of algorithm 3
  else
  continue with the next iteration of the current loop
  endif
16 **endwhile**
17 **execute** the loop of algorithm 3

Algorithm 2: Binary search on the *current-search-sector*

1 {Each iteration on the divided sectors}
2 **if** the *current-question-range* intersects sectors in lower half **and** the maximum value of any sector in lower half is $\geq V$ **then**
3 **if** *current-question-range* also intersects sectors in upper half **then**
4 make a copy record of the current values, i.e. *current-question-range*, *next-match*, *current-tree-level* and *partition-key* which are to be restored later if back tracking is required
endif
5 continue the binary search in the lower half
6 **else**
7 **if** *current-question-range* intersects the upper half **then**
8 continue the binary search in the upper half
else
9 **if** no copy record exists **then**
10 return FALSE
{No *next-match* exists, question process terminates}
else
11 restore copy record of variables with those recorded
continue the binary search in restored search sector
endif
endif
endif

Algorithm 3: Finds a *next-match*, restricted to a sector

{The data points are restricted to a sector in which their values are greater than the *partition-key* and a *next-match* exists}
1 **while** *current-tree-level* is lower than or equal leaf level **do**
2 **if** the *current-question-range* = *current-sector* **then**
all the digit values for upper levels $\Leftarrow$ 0
return TRUE
{The *next-match* has been identified}
endif
3 find in which sector of the *current-search-sector* the coordinate boundaries of *current-question-range* lies
4 perform a binary search to find the *current-sector* which intersects with the *current-question-range*, whose the value is minimum
5 **if** *current-tree-level* $\neq$ the leaf level **then**
current-question-range $\Leftarrow$ *current-question-range* $\cap$ *current-sector*
current-search-sector $\Leftarrow$ *current-sector*
endif
6 *next-match* $\Leftarrow$ *next-match* + ((the value of current-sector) << (m (*current-tree-level*))
{Append the value of *current-sector* to the *next-match* }
7 *current-search-sector* $\Leftarrow$ *current-sector*
8 *current-tree-level* $\Leftarrow$ *current-tree-level* + 1
endwhile
9 return TRUE

In what follows, we explain the data retrieval technique for such data points mapped into the Hilbert curve using an example in figure 3. Here we examine for local computation of partition $P3$ where a data point $X1$ generates a question region. The data retrieval process proceeds as follows:

1. Define a question range. The points $Q1$ and $Q2$ define a question range generated by point $X1$.
2. The `partition-key` is initialized as the lowest value of partition $P1$. The `exa-partition` is partition $P3$.
3. The **findNextMatch** function is called and it determines the lowest point on the Hilbert curve to the question range. This value is assigned to the `next-match` (in this case, it is the point M).
4. This point is searched and it is found in partition $P2$, which the lowest value is B'.
5. Partition $P2$ is currently searched and the data points C, B and A are found in the question range and they are retrieved.
6. Since $P3$ is the `exa-partition`, it is skipped. The `partition-key` is then set into $P4$.
7. The **findNextMatch** function is called and it determines the lowest value of $P4$, which is D'. Set this value to the `next-match`.
8. Partition $P4$ is now searched and there are no data points that lie in the question range.
9. Following the one just searched, the `partition-key` is then set into $P5$.
10. The **findNextMatch** function is called to determine the lowest value of $P5$. This value is E', which then assigned into `next-match`.
11. Partition $P5$ is now searched and there are no data points that lie within the question

range.

12. The `partition-key` is then set to the partition $P6$, following the one just searched.
13. The **findNextMatch** function is called and determines the lowest value of $P6$ and it is F' . Set this value to the `next-match`.
14. Partition $P6$ is now searched and the data points D, E, F and G are found which lie within the question range and they are retrieved.
15. The `partition-key` is then set to the partition $P7$.
16. The **findNextMatch** function is called to determine the lowest value of $P7$ and it is G' . Set this value to the `next-match`.
17. Partition $P7$ is currently searched and the data points H and I are found that lie on the question range and they are retrieved.
18. The `partition-key` is now set to $P8$, following the one just searched. ($P8$ is the final partition).
19. Determine the lowest value of $P8$ and it is H' . Set this value to the `next-match`.
20. Partition $P8$ is now searched and the data points J, K and L are found which lie within the question range and they are retrieved. Since there is no higher partition that to be searched, the data retrieval process therefore terminates.

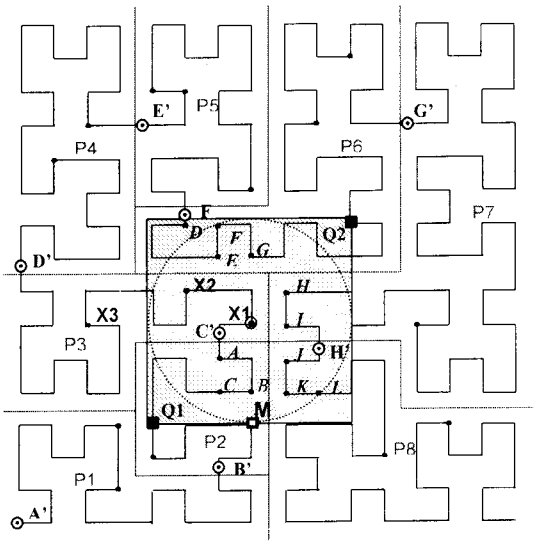

Figure 3: A retrieval process on the Hilbert curve

The method for finding a `next-match` is an iterative process on the divided sectors. The `current-search-sector` is initialized as the original sector, the root node, which has label 0. At each iteration process we search a sector or a node in `current-search-sector`, which is the parent of a node in which the search continues in the next iteration. The search is restricted on the `current-search-sector`, which intersects with the question range till a `next-match` is found in a leaf node. Therefore the search process corresponds to descending the tree structure. The following is an example on finding the lowest value on the Hilbert curve for the question range (point M in this example).

Tree level 1: The `current-search-sector` is the root node, which has label 0. The order of its sub-nodes is 00, 01, 02 and 03. The question range is defined by Q1 and Q2. The lower two sub-sectors, labeled 00 and 01 intersect with the question range, and so the search starts at these two sectors restricting sub-sectors labeled 02 and 03. Since the question range intersects both these sub-sectors, `next-match` is tentatively set as 00.

Tree level 2: The `current-search-sector` is now restricted to that labeled 00. The order of its sub-sectors is 000, 001, 002 and 003, which are the labels of sub-sectors of level 2. The lower two sub-sectors, labeled 000 and 001 intersect with the question range. Thus two sub-sectors, labeled 002 and 003 could be neglected and the search continues to the sector 000. Since the question range does not intersect with this sector, `next-match` is tentatively set as 000. The search backtracks to the sector of label 001 and so `next-match` is updated to 001.

Tree level 3: The `current-search-sector` is now restricted to the sector that is labeled 001. The order of the labels of its sub-sectors is 0010, 0011, 0012 and 0013. Since the sub-sectors that intersect with the question range are of label 0012 and 0013, the search neglects two lower sub-sectors label 0010 and 0011 and continues to the sector 0020. Since the question range intersects this sector, `next-match` is tentatively set as 0020.

Tree level 4: The `current-search-sector` is now restricted to the sector labeled 0020. The labels of its sub-sectors are in the order of 00200, 00201, 00202 and 00203 and are the leaf sectors. The search neglects two lower sub-sectors 00200 and 00201 and continues to sector 00202. Therefore `next-match` is then set as 00202, which is the lowest value M to the question range.

5. Complexity of the Algorithms

The Hilbert curve construction process is equivalent to the tree building procedure. This divides a sector containing many data points into four and uses the pointers to link the divided sectors in a tree structure. This process continues till each node or sector can not have plural data points. At that point, the sector decomposition ends and hence, the height of the corresponding tree is finite. The order of the curve equals the height of the tree. The complexity of such tree construction algorithm is determined by the height of the tree, which is the order of the curve k or equivalently the number of digits required to label the divided sectors. Each smallest sector enclosing a point at the level of decomposition k is visited once during each recursion process and the complexity is determined by the number of n, which is the number of the given data points. Thus the overall complexity of the Hilbert curve construction algorithm can be stated as $O(kn)$. In the case of [9], more than one works are required for rotating the divided sectors in every step of recursion even though the order of complexity is the same.

For the data retrieval algorithms, the complexity is determined by a binary search, which is done on a sorted linear list of the data points. Back tracking to the lower level of a particular level is required but not always necessary and so the complexity of each iteration process remains $O(n)$.

6. Conclusions

We have described a sector labeling method for the Hilbert space-filling curve construction. The method leads to a robust approach, which manages to avoid complicated bit interleaving operations as well as wasteful sector rotations, with certain flexibility to handle regular and irregular data points.

This method leads into the excellent idea on developing data retrieval algorithms for the data points mapped into the Hilbert curve. The technique is straightforward and the algorithms make it easy to decide which partitions contain the data points that meet specific criteria and must be retrieved for the local computation of a partition. This is useful in many scientific applications such as a many-body simulation.

Furthermore, the underlying technique is general. With a little modification, it can be applied for partitioning the data points that based on the other space-filling curves such as the Z-curve. In fact, the technique has the capability to determine the data points that are essential for local computation of a partition, from which therefore, one may estimate the communication effort between processors. The data retrieval algorithms can also be used for an important benchmark in the field of scientific simulations.

References

[1] Schlick, T., Skeel, R., Brunger, A., Kale, L., Board Jr, JA., Jermans, J. and Schulten, K., "*Algorithmic Challenges in Computational Molecular Biophysics*", J. Comp. Physics, Vol. 151, pp. 9-48, 1999.

[2] Hendrickson, B. and Devine, K., "*Dynamics Load Balancing in Computational Mechanics*", Comp. Methods Appl. Mech. Engrg., Vol. 184, pp. 485-500, 2000.

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

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

[5] Hernquist, L., "Hierarchical N-Body Methods", J. Comp. Phys. Comm., Vol. 48, p. 107-115, 1988.

[6] Berger, MJ., and Bokhari, SH., "*A Partitioning Strategy for Non-uniform Problems on Multiprocessors*", IEEE Trans. Comp., Vol. C-36, p. 570-580, 1987.

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

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

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

[10] Pilkington, JR., and Baden, SB., "*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.

[11] Eric Jui-Lin Lu, "*An Efficient Load Balancing Technique for Parallel FMA in Message Passing Environment*", Proceedings of Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.

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