A study on fast multipole algorithms and data structures

Dr. Anda-CSCI 694 Fall 2008

12/16/2008

St. Cloud State University
Mool Amit & Syed Zafrul

Contents

Table of figures

Introduction to the FMM

The fast multipole algorithms were introduced in order to approximate the gravitational N-Body problems with pairwise interactions. The FMM were then extended to take into the account the Laplace equations which are solved using integrals. Recently the FMM was applied to solution of Maxwell equations using finite element time domain algorithms.

The FMM has also been applied in accelerating the iterative solver in the method of moments (MOM) as applied to computational electromagnetics problems. The FMM was first introduced in this manner by *Greengard* and *Rokhlin* and is based on the multipole expansion of the vector Helmholtz equation. By treating the interactions between far-away basis functions using the FMM, the corresponding matrix elements do not need to be explicitly stored, resulting in a significant reduction in required memory. If the FMM is then applied in a hierarchical manner, it can improve the complexity matrix-vector product in an iterative solver from $O(N^2)$ to $O(N \log N)$. This has expanded the area of applicability of the MOM to far greater problems than were previously possible.

The FMM, introduced by Rokhlin and Greengard, has been acclaimed as one of the top ten algorithms of the 20th century. The FMM algorithm dramatically reduces the complexity of matrix-vector multiplication involving a certain type of dense matrix, which can arise out of many physical systems.

n-Body Problem

The n-body problem is the problem of finding, given the initial positions, masses, and velocities of n bodies, their subsequent motions as determined by classical mechanics and Newton's laws of motion.

Some Examples

Some examples of n body problems include Bio-physics ion conduction , interaction in a planetary system in astrophysics, atomic simulation in bio medicine, one of the smallest simulated system of real size biology problem has about 2^{15} molecules.

Basic Idea of FMM

The basic idea of FMM is that instead of separately interacting with each point that is at a certain distance , a particle outside the circle area can interact with an approximation of the distant group; this means that the points in the circle are far enough away that they can be considered as a single point .

The idea is to repeat this for the entire system ; the points that are not far enough are considered as close range points and we must therefore define a way to compute their interactions.

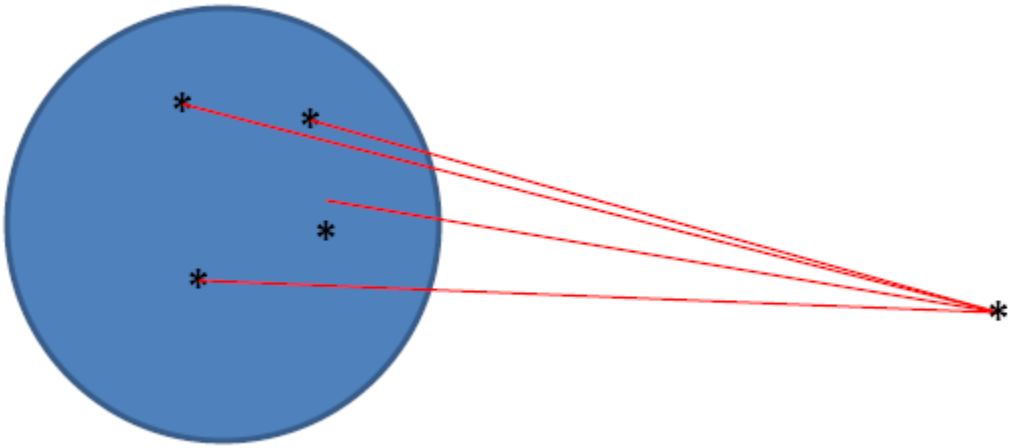

Figure 1 Interaction with points: case 1

The figure above shows the way we interact with points if FMM is not used, and the figure in the following page shows the FMM way of interacting with the same points, which dramatically reduces the number of interactions.

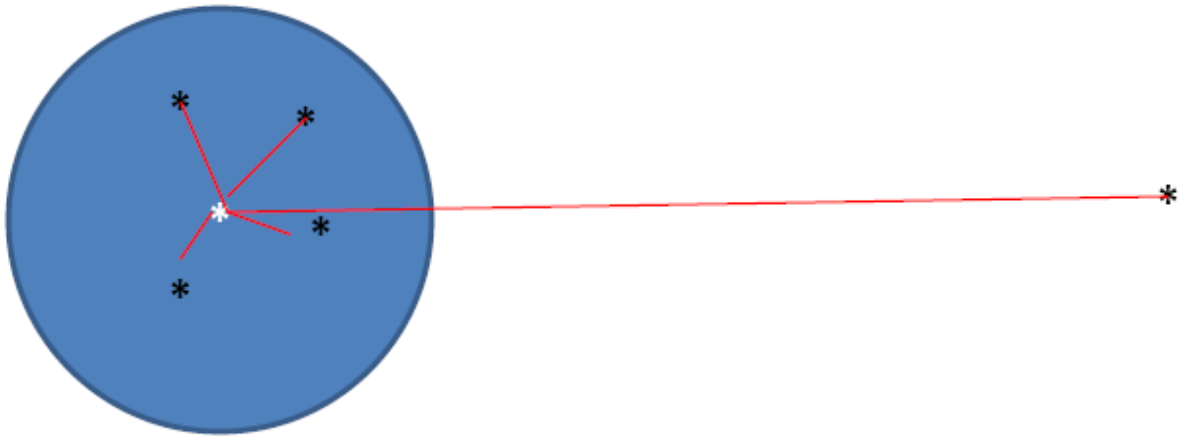

Figure 2 Interaction with points: Case 2

Motivation for the FMM

The main motivation behind FMM is to improve on the O(N^2) complexity problem. In this process this became one of the top ten algorithms in the century.

All the problems discussed as a part of the examples for n-body problem are main reasons for motivation for the FMM.

The other important idea is the hierarchical division of space so that the simulation regions are suitably separated and distant from each other in order to maximize aggregated group interactions instead of single particle ones. A structure that is particularly adapted to this space decomposition process is the octree in 3D or quad-tree in 2D.

Memory complexity issues

At times we are not able to solve a problem in available memory. We don't care how long the solution takes and we are just worried about the solution. For example to store a N X N matrix we need N^2 locations.

For instance if we consider a machine with 1GB RAM which is equivalent to 1024^3 which is equal to 1.073,741,824 bytes. And in this case the largest possible is 32,768.

Multi Level FMM

The multi level FMM provides a generalized approach of the FMM in d dimensions. Here the data structure used is octree. Focus is laid on how the points are aggregated in optimal clusters for efficient computations and also find ways to improve matrix vector product computations.

Octree

An octree is a tree data structure in which each internal node has up to eight children. Octrees are most often used to partition a three dimensional space by recursively subdividing it into eight octants. Octrees are the three-dimensional analog of quadtrees. The name is formed from *oct* + *tree*, and normally written "*octree*", not "*octtree*".

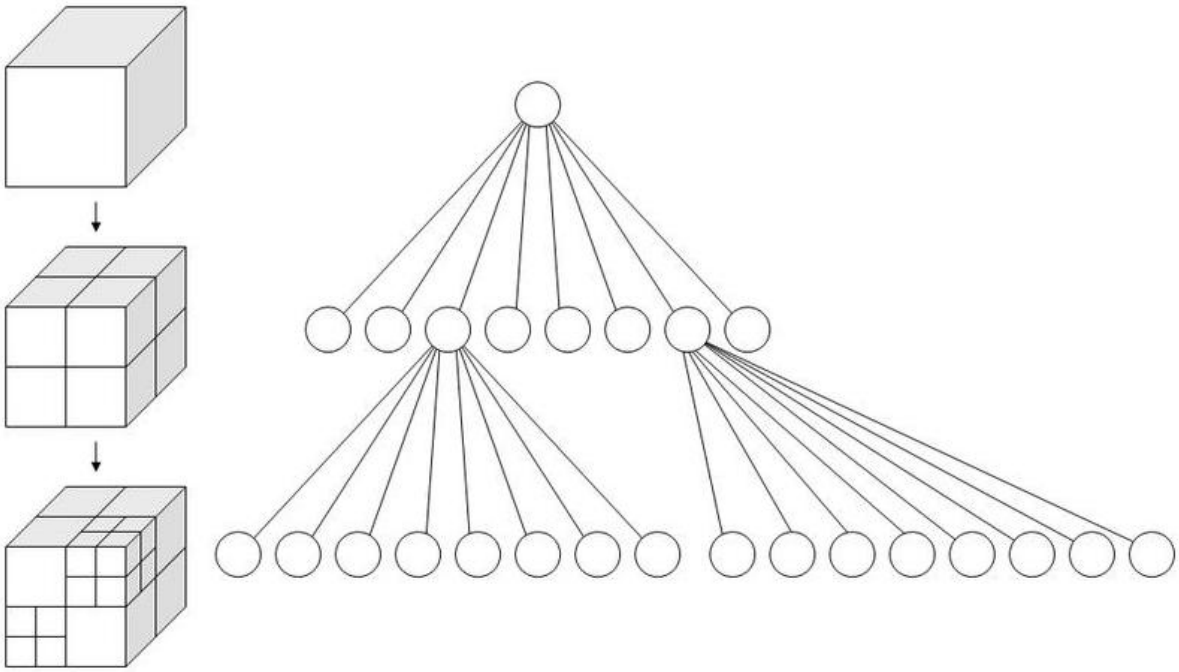

Figure 3 Octree , A tree data Structure

Each node in an octree subdivides the space it represents into eight octants. In a point region (PR) octree, the node stores an explicit 3-dimensional point, which is the "center" of the subdivision for that node; the point defines one of the corners for each of the eight children. In an MX octree, the subdivision point is implicitly the center of the space the node represents. The

root node of a PR octree can represent infinite space; the root node of an MX octree must represent a finite bounded space so that the implicit centers are well-defined. When used in this way, octrees are a special case of k-dimensional trees, in which the coordinate system is three dimensional.

Working of MLFMM

The working of MLFMM is better understood by examining an example of Matrix vector product:

$$s(x_j) = \sum_{i=1}^{N} \alpha_i \phi(x_j - x_i), \quad \{s_j\} = [\Phi_{ji}]\{\alpha_i\}.$$

Direct evaluation of this product requires O(*MN*) operations. In the FMM, we assume that the functions which constitute the matrix can be expanded as local series or multipole series that are centered at locations y* and x* as follows:

$$\phi(y) = \sum_{q=0}^{p-1} a_q(y_*) R_q(y - y_*) + \varepsilon(p), \qquad \phi(y) = \sum_{q=0}^{p-1} b_q(x_*) S_q(y - x_*) + \varepsilon(p)$$

Middleman Method

The middleman method which is applicable only to the functions that can be represented by a single uniform expansion everywhere(Say an expansion in terms of R basis above).In this case we can perform the simulation efficiently by first performing a p-term expansion for each of the N functions, $\emptyset_i$ at a point x* in the domain requiring *O(Np)* operations. Similarly *Mp* operation when we dela with M functions , puttogethr we can obtain a *O(Mp+Np).*

The following pictures give a pictorial demonstration of Middle man algorithm.

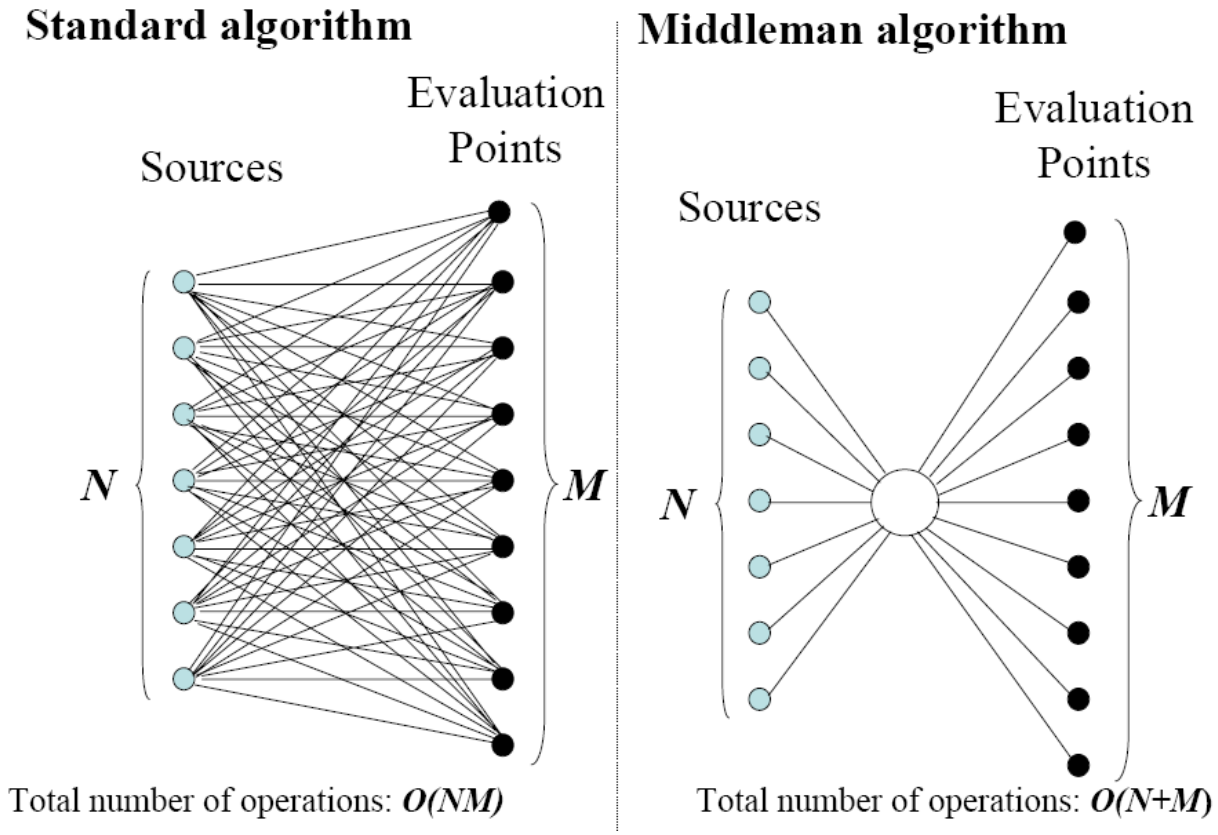

Figure 4 Middleman Algorithm

FMM groups and translates the approximations that each source generates in order to reduce the asymptotic complexity.FMM tries to achieve a complexity of O(M+N) or O(M+NlogN) by factorization and translation properties of functions. The original FMM uses 2-d tree space division**.** The assumed translational operators would look like this

$$a_q(y_{*_2}) = (R \mid R)(y_{*_2}, y_{*_1})[a_r(y_{*_1})],$$

$$b_q(x_{*_2}) = (S \mid S)(x_{*_2}, x_{*_1})[b_r(x_{*_1})],$$

$$a_q(y_*) = (S \mid R)(y_*, y_*)[b_r(y_*)],$$

The pictorial demonstration of multilevel FMM is shown below. Where the (R|R), (R|S) and (S|S) are the translational operators that allow the transformation of the coefficients between different bases.

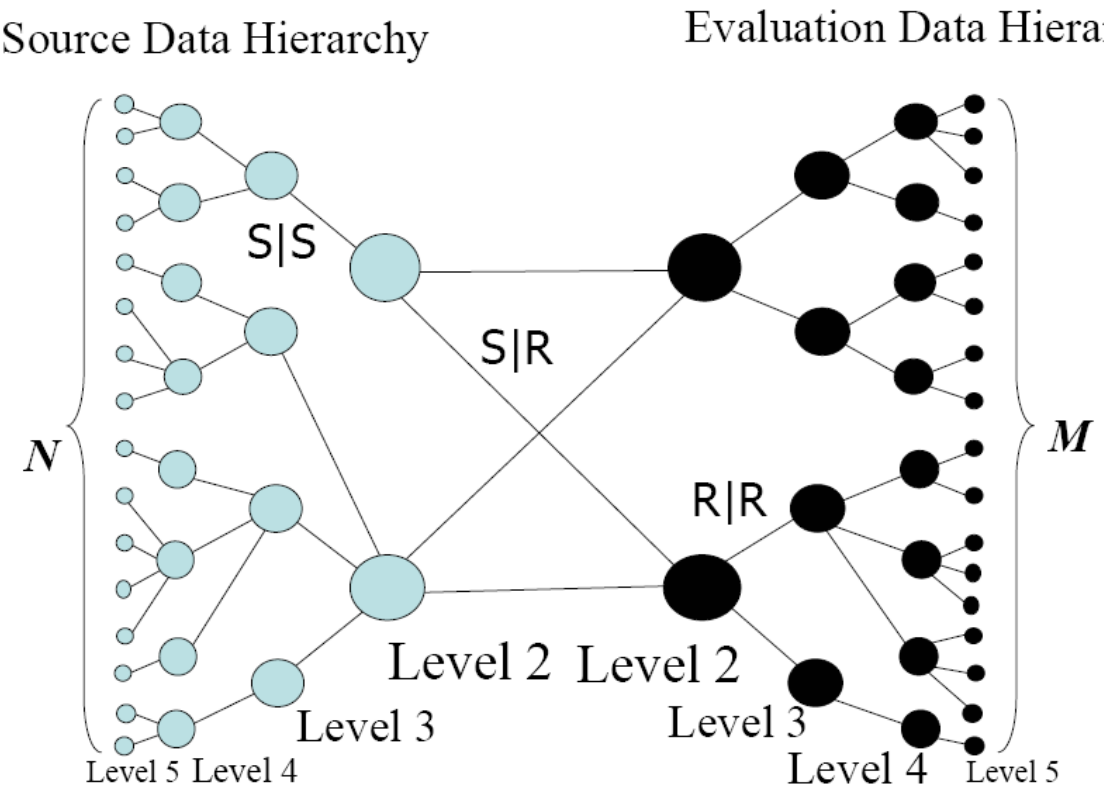

Figure 5 Multi Level FMM

FMM Data Structures:

The fast multipole method allows the rapid approximate evaluation of sums of radial basis functions. The method scales as O(N) in both time and memory compared to the direct method with complexity $O(N^2)$, which allows the solution of larger problems with given resources. This system uses the hierarchical space subdivision with 2^d- Tree. It maintains the parent child relationship using the indexing system. The structure of the FMM can be implemented using either the Tree structure or using Box structure. FMM can also be used on Sets and Spatial ordering using Bit interleaving.

The FMM can be implemented using Quadtrees (2D) or Octrees(3D). The 2^d-tree structure can be presented as below

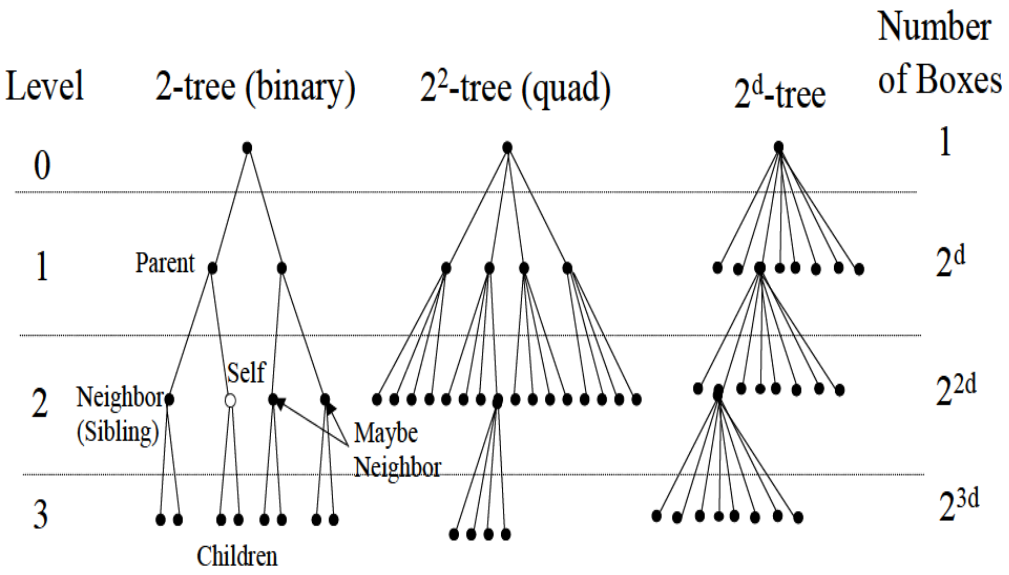

Figure 6 2d-tree structure

Here d represents which type of tree is used. 2^d at any level specifies the number of child a node can have. But for our presentation we had used the box structure in which a single box is symmetrically sub divided in a sub-box where each sub section represent the child node. It can be pictorially shown as below

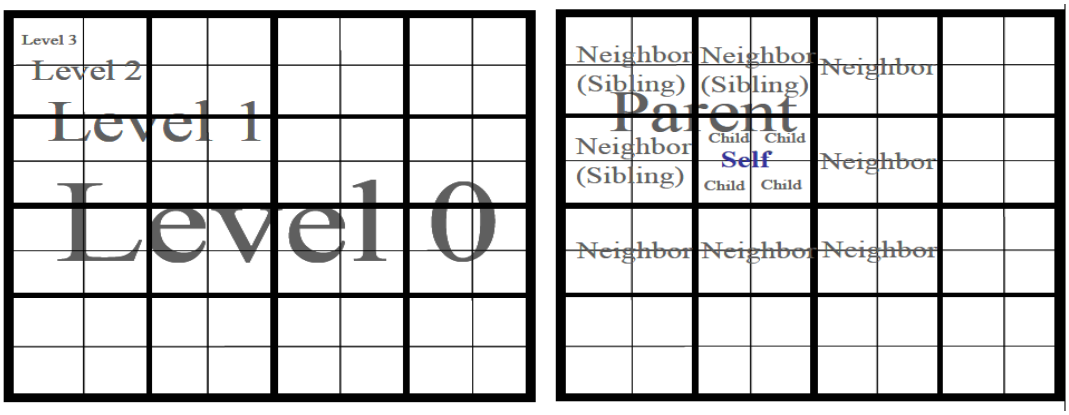

Figure 7 Box representation of 2d-tree structure

If we take the whole box it will act as at the root level, the sub-division and the specific naming of each sub-box has been shown in the right side diagram.

Universal and Hierarchical Indexing in 2^d-trees

The indexing for the Quad tree can be demonstrated as shown in the figure below.

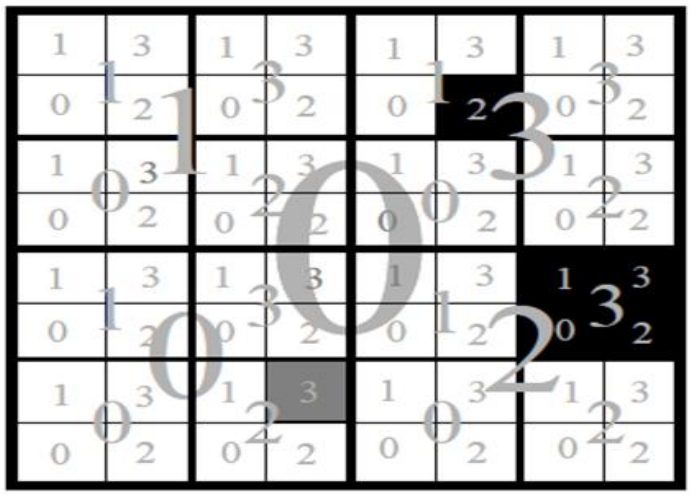

Figure 8 Indexing of Quad Tree

For our example we will consider the black and grey boxes. The number in the above box represents its respective position and the square dark lines separate each box and its level. The large black box has a index of (2, 3) which means 2 is the parent and 3 is the child on which we are focusing our interest. Since we have taken the Quadtree it base will be 4 so when $(2, 3)_4$ is converted to decimal system its index position will be 11. The index for the small black box is (3, 1, 2) and is particular number is 2. Its grandparent is 3 and parent is 1. So its index will be 54 in decimal format. To find the index of a particular box in hierarchical order we can use the expression

$$Index\ number = (2^d)^{l-1} * N_1 + (2^d)^{l-2} * N_2 + \ldots + (2^d)_{l-1} * N_{l-1} + N_l$$

Here d is the type/order of tree used and l is the level of the box in the entire tree.Similarly the Universal index number for the large black box is (2, 3) and its decimal format index is 11. The universal index for the small grey color box is given by (0, 2, 3) which is given as 3 is its position index, 2 is its parent and 0 is the grandparent which is at the level 1. So its index 23_4 is equal to 11.

The universal index pair can be given as

$$UniversalNumber = (Number, l)$$

Parent, Children and neighbor Index

The parent of each node can be easily calculated using the expression

$$Parent(Number) = (2^d)^{l-2} * N_1 + (2^d)^{l-3} * N_2 + ... + N_{l-1}$$

Where the parent indexing string is Parent($N_1, N_2, ... N_{l-1} N_l$) = ($N_2, N_2, ... N_{l-1}$). Likewise the universal index for the parent can be calculated as *Parent(Number, l) = (Parent(Number), l-1).*

The children universal indexes can be calculated as Children(Number, l) = (Children(Number), l+1) as the children indexing string is calculated as $Children(N_1,N_2,..,N_{l-1},N_1) = \{(N_1,N_2,..,N_{l-1},N_1, N_{l+1})\}$, $N_{l+1} = 0,...,2^d-1$.

To find the parents and children of subsequent nodes we can use the algorithm as

$Parent(Number) = [Number/2^d]$ *and*

$Children(Number) = \{ 2^d * Number + j\}, j = 0,...,2^d-1$

So in figure 3 the parent of the box #23 has the index of 2.

The neighbor of any node at level l is equal to *Neighbor((Number, level))=Number±1* but we will drop from the neighbor lis if it is equal to 2^l or -1.

Bit Interleaving and Deinterleaving:

We use the FMM if the number of the items are more than or equal to 10^4, it is suitable for the n-body sample items. To minimize the number of memory usage we will use the interleaving and deinterleaving technique. Initially the bits can be represented as

$X = (0.b_{11}b_{21}..b_{d1}b_{12}b_{22}..b_{d2}...b_{1j}b_{2j}..b_{dj}..)$.

We will mix different bits of the number. A unique data can be generated using different combination of numbers arranging them in a horizontal order. For example

$X= (0.N_1N_2N_3…N_J…)_2^d$, $N_j = (b_{1j},b_{2j}…. b_{dj})_2$, j=1,2,…., Nj = 0,….2^d -1 .

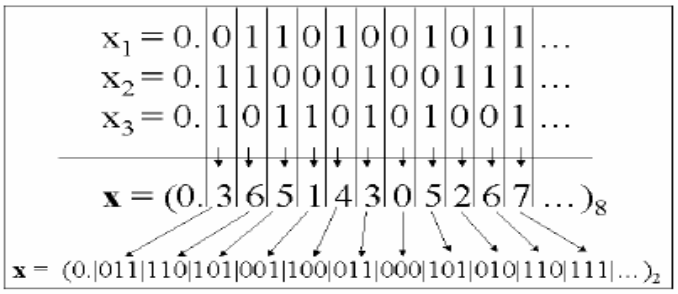

Figure 9 Bit Interleaving and Deinterleaving

Similarly to deinterleave the certain number first select the the fix number of bits which will be used to generate how many different numbers are needed once deinterleaving is done. For example we have a number 76893 its binary equivalent will be 10010110001011101. If we use a combination of 3 bits to generate the three different numbers we will select each group of bits and start allocating each bits to a numbers starting from the left side. Here, the position of the number is fix and will not change. The left most bits of the original string will appear on the left side of each numbers subsequently. The figures below will demonstrate the actual working pattern of bit deinterleaving.

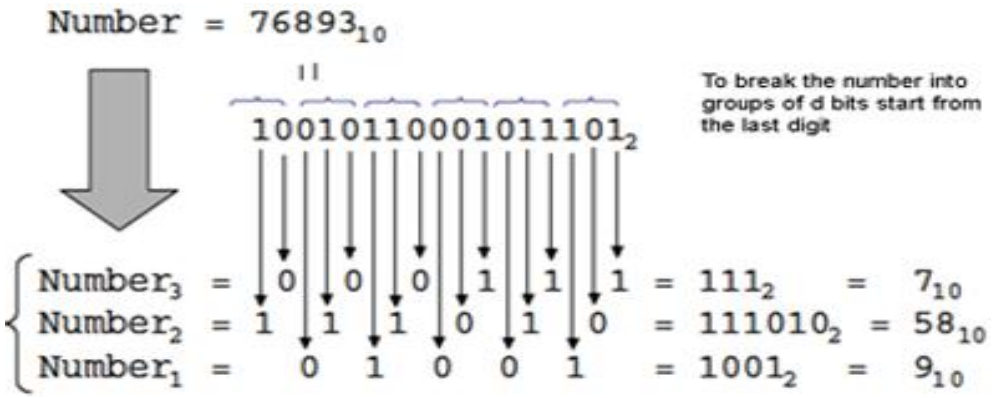

Figure 10 working pattern of bit deinterleaving.

Now, to find the subsequent neighbor we can use the expression as

$$Sk = \{ Nos^{-}_k, Nos_k\, Nos^{+}_k\}, \quad Nos_k \neq 0,\ 2^l - 1$$

$$\{ Nos^{-}_k, Nos_k\, Nos^{+}_k\}, \quad Nos_k = 0,\ k = 1 \ldots d$$

$$\{ Nos^{-}_k, Nos_k\, Nos^{+}_k\}, \quad Nos_k =,\ 2^l - 1$$

We will convert the numbers we get into binary format and apply the interleaving on them gives us the neighbor of a particular k node. For example to find the neighbor of a node containing data 26

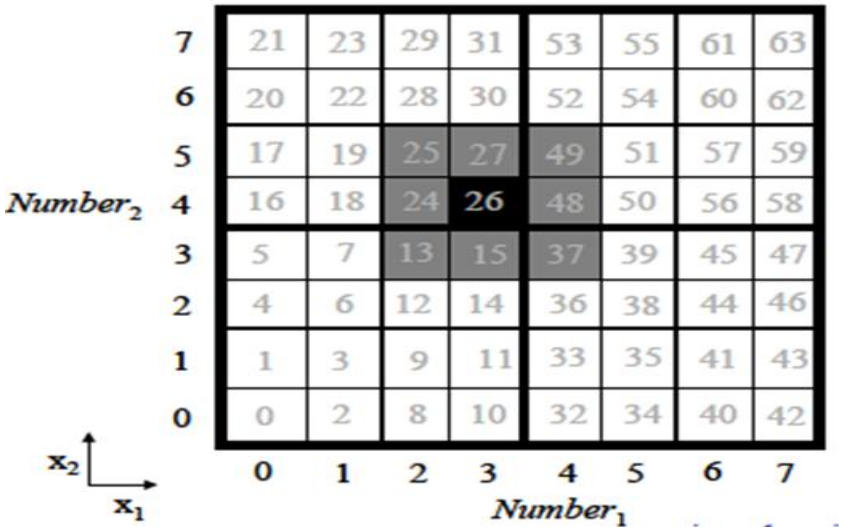

Figure 11 Finding Neighbor

we will convert it to binary number and use the deinterleaving technique which will give use the different numbers depending on the size of a data on a particular node. Here 26_{10} is equivalent to 11010_2 . We get two different numbers $(11, 100)_2 = (3, 4)_{10}$. Now we use the above mentioned S_k function to calculate tis different neighbors. The result is (2,3), (2, 4), (2, 5), (3, 3), (3, 5), (4, 3), (4, 4) and (4, 5).

These all individual numbers are converted to the binary numbers which will be (10, 11), (10, 100), (10, 101), (11, 11), (11, 101), (100, 11), (100, 100), (100, 101).

Implementing interleaving in all these sequence of number will generate 1101, 11000, 11001, 1111, 11011, 100101, 110000 and 110001 which is equivalent to 13, 24, 25, 15, 37, 48, 49.

This is the same result that had been shown in the figure 11.

References:

- Duraiswami & Gumerov, 2003-2004, FMM CMSC 878R/AMSC 698R, http://www.umiacs.umd.edu/~ramani/cmsc878R/2004/Lecture11.pdf
- Nail A. Gumerov *, Ramani Duraiswami, 2008, Fast multipole methods on graphics processors, http://www.umiacs.umd.edu/~ramani/pubs/Gumerov_Duraiswami_GPUFMM_JCP_2008.pdf
- Felipe A. Cruza, L. A. Barbaay and Matthew G. Knepleyb, 2008, Fast Multipole Method for particle interactions: an open source parallel library component, http://www.maths.bris.ac.uk/~aelab/files/abstracts/CruzBarbaKnepley-PCFD08.pdf
- Çakir, Özcan, 2006, The multilevel fast multipole method for forward modelling the multiply scattered seismic surface waves, volume 167 http://www.ingentaconnect.com/content/bsc/gji/2006/00000167/00000002/art00019