Fast Multipole Algorithms and Data Structures

CSCI 694

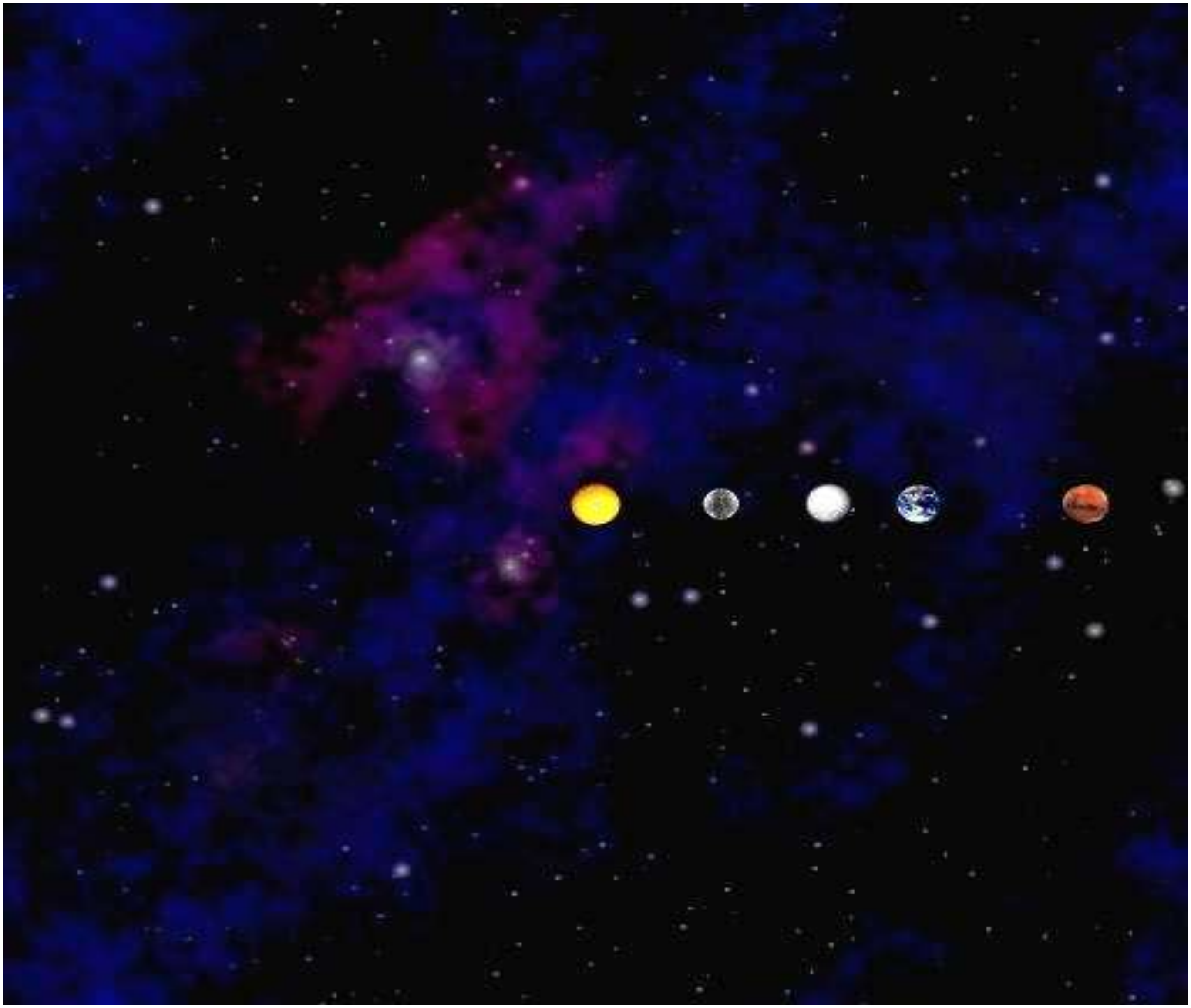
Dr.A.Anda

Fall 2008

Mool Amit
Syed Zafrul

Fast Multipole Method

A mathematical technique to speed up the calculation of long ranged forces in ***n –body problem***

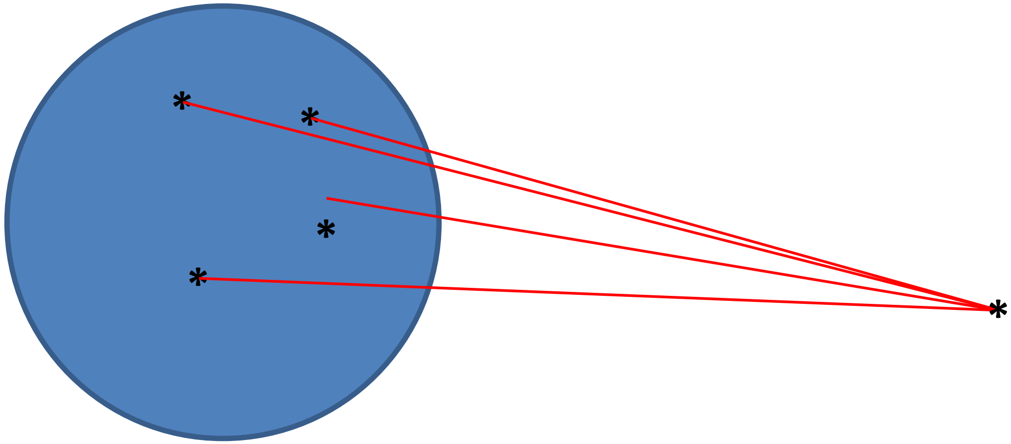


Understanding n-body problem

Some examples

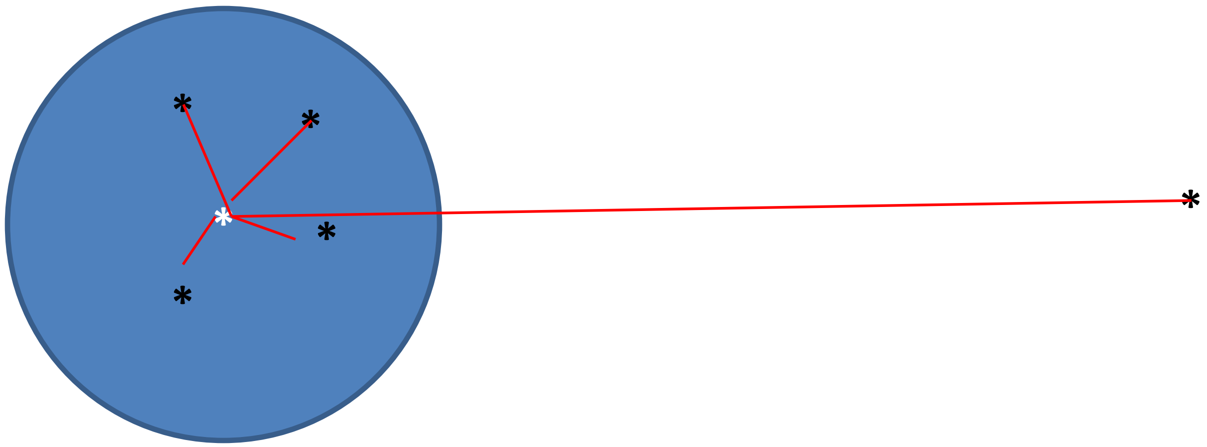
- Bio Physics- ion conduction
- Interaction in a planetary system in astrophysics
- Atomic simulation in bio medicine
- One of the smallest simulated system of a real size biology problem has 2^{15} molecules.

Basic FMM Idea



Instead of separately interacting with each point charge that is at a certain distance…………

Basic FMM Idea..contd



A particle outside the circle can interact with an approximation of a distant group, this means that the particles in the circle are far enough that they can be considered a single point

Why FMM ?

Growing importance of Computational simulation

Many simulations involve several million variables

- Most large problems boil down to solution of linear system or

  performing a matrix-vector product

- Regular product requires *O(* N^2*) time and O(* N^2*) memory*
- The FMM is a way to accelerate the products of particular

  dense matrices with vectors

Do this using *O(N) memory*

- FMM achieves product in *O(N) or O(N log N) time and memory*

Where FMM can be applied ?

The need for fast algorithms

- Grand challenge problems in large numbers of variables
- Simulation of physical systems
    - Stellar clusters
    - Protein folding
    - Acoustics
- Graphics and Vision
    - Light scattering …

Memory complexity issues

Memory complexity

- Sometimes we are not able to fit a problem in available memory

  Don’t care how long solution takes, just if we can solve it

- To store a *N × N matrix we need N2 locations*

  1 GB RAM = 10243 =1,073,741,824 bytes

  => largest *N is 32,768*

- FMM allows reduction of memory complexity as well

Elements of the matrix required for the product can be generated as needed

  Can solve much larger problems (e.g., 107 variables on a PC)

FMM.. Introduced by Rokhlin & Greengard in 1987

- Called one of the 10 most significant advances in computing of the 20th century
- Speeds up matrix-vector products (sums) of a particular type

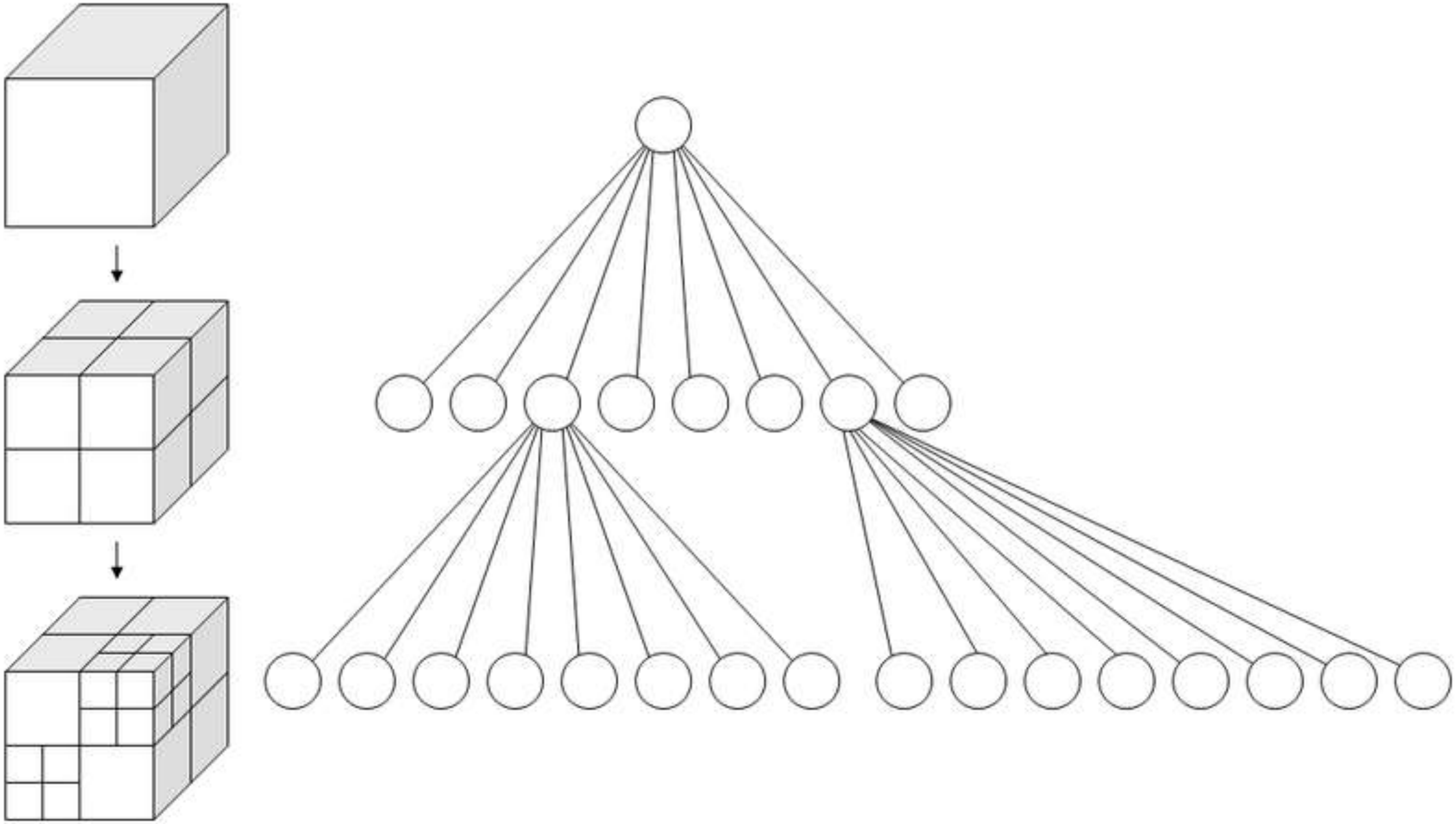
$$s(x_j) = \sum_{i=1}^{N} \alpha_i \phi(x_j - x_i), \quad \{s_j\} = [\Phi_{ji}]\{\alpha_i\}.$$

- Above sum requires *O(MN) operations.*
- For a given precision *ε the FMM achieves the evaluation in O(M+N)* Operations.

Multi Level FMM

- Gives a generalized approach of the FMM in d dimensions
- Data structure used is octree
- Focus is on how the points are aggregated in optimal clusters for efficient computation

Octree



Working of MLFMM

Consider the following matrix vector product

$$v_j = v(y_j) = \sum_{i=1}^{N} u_i \phi_i(y_j), \quad j = 1,\ldots,M, \quad [\phi]\{u\} = \{v\}.$$

Direct evaluation of the product requires O(MN) operations

Assumption of FMM is that the functions which constitute the matrix can be expanded as

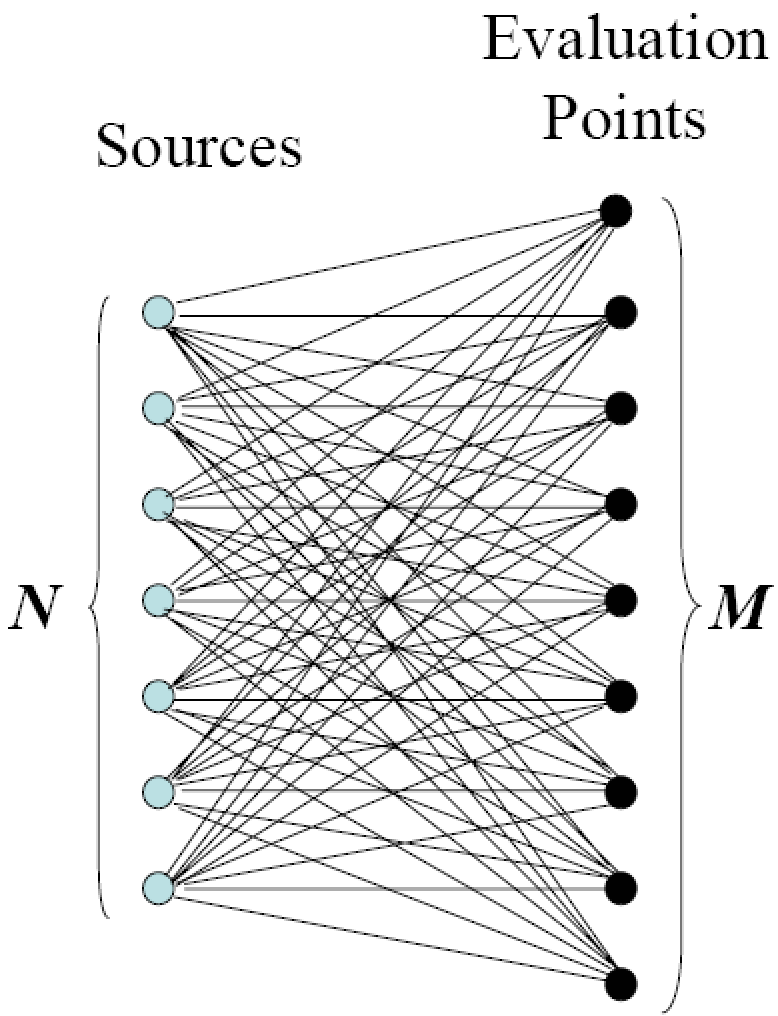
Local series and

Multipole series at locations y_* and x_*

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

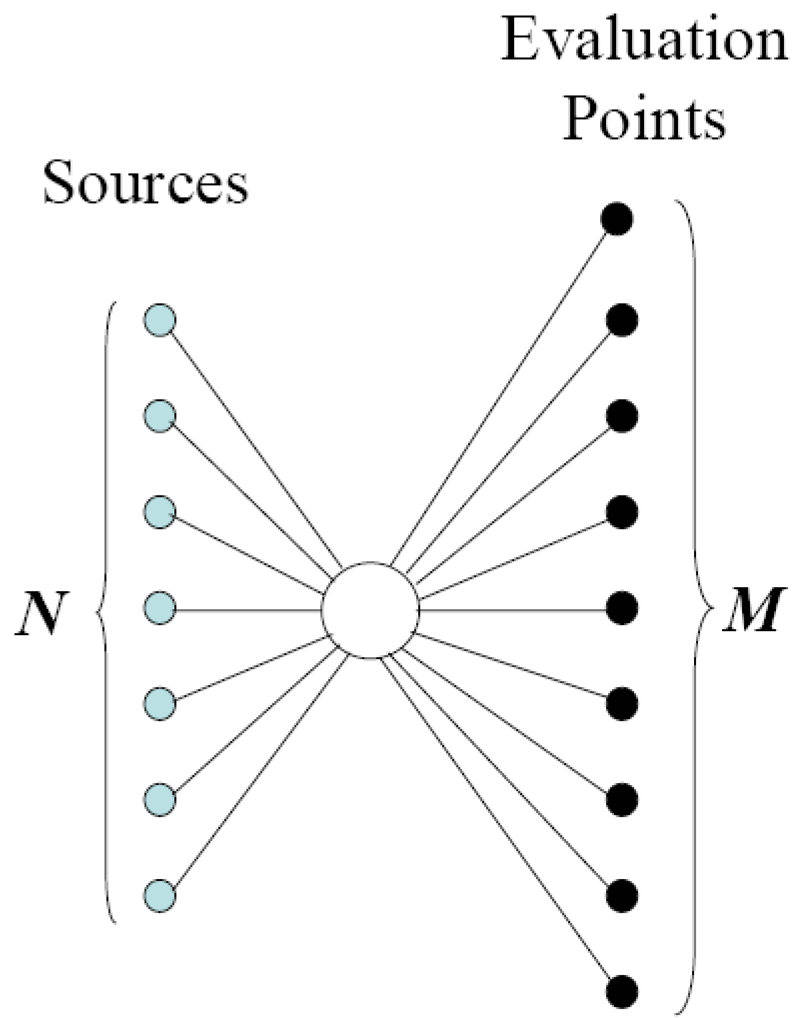
MIDDLEMAN Method

Standard algorithm



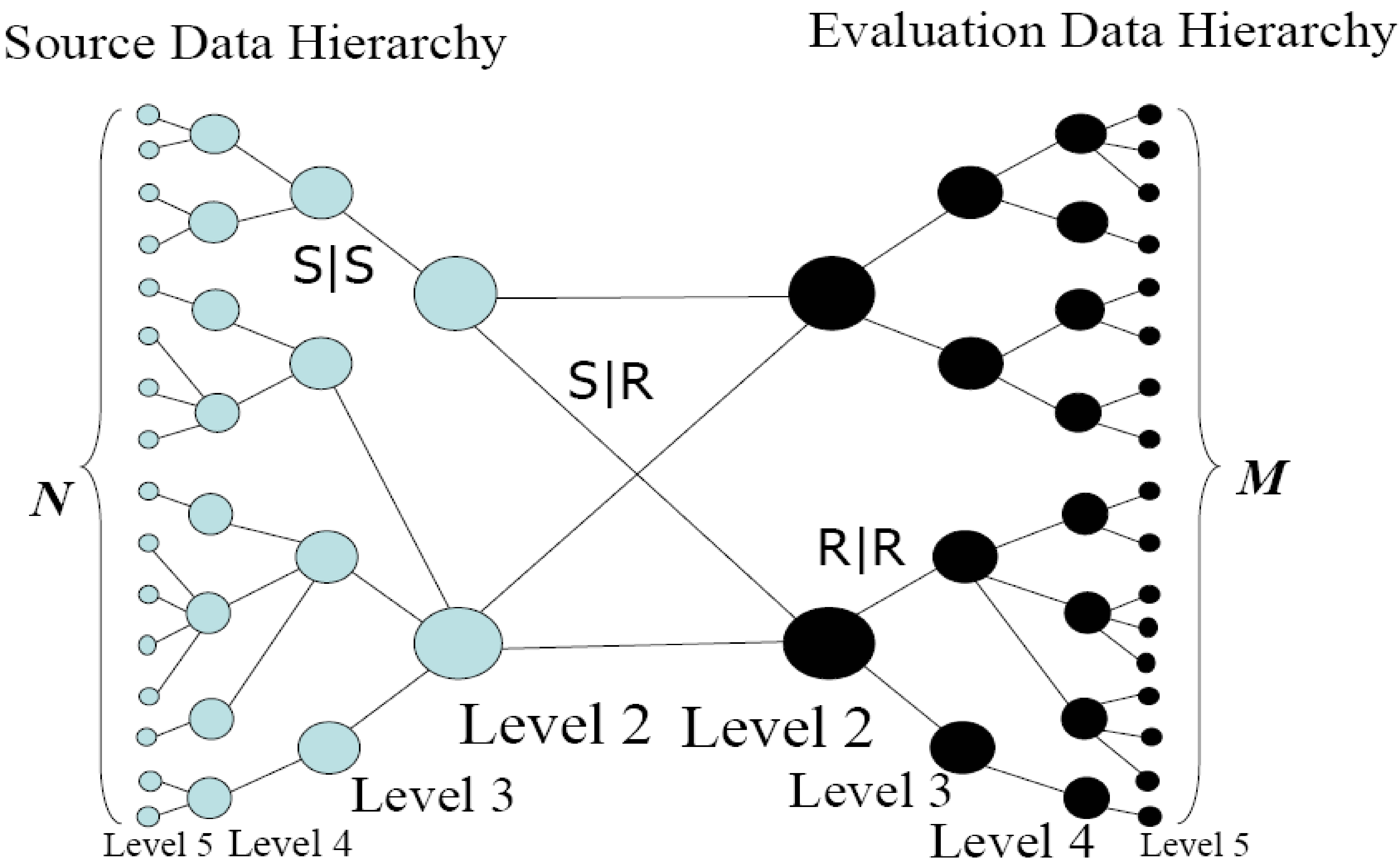
Total number of operations: $O(NM)$

Middleman algorithm



Total number of operations: $O(N+M)$

MLFMM


Source Data Hierarchy
Evaluation Data Hierarchy
S|S
S|R
R|R
N
M
Level 2
Level 2
Level 3
Level 3
Level 5
Level 4
Level 4
Level 5

MLFMM.. In short

- 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 $O(M+N\log N)$ by factorization and translation properties of functions.
- The original FMM uses 2-d tree space division.

FMM Data Structures

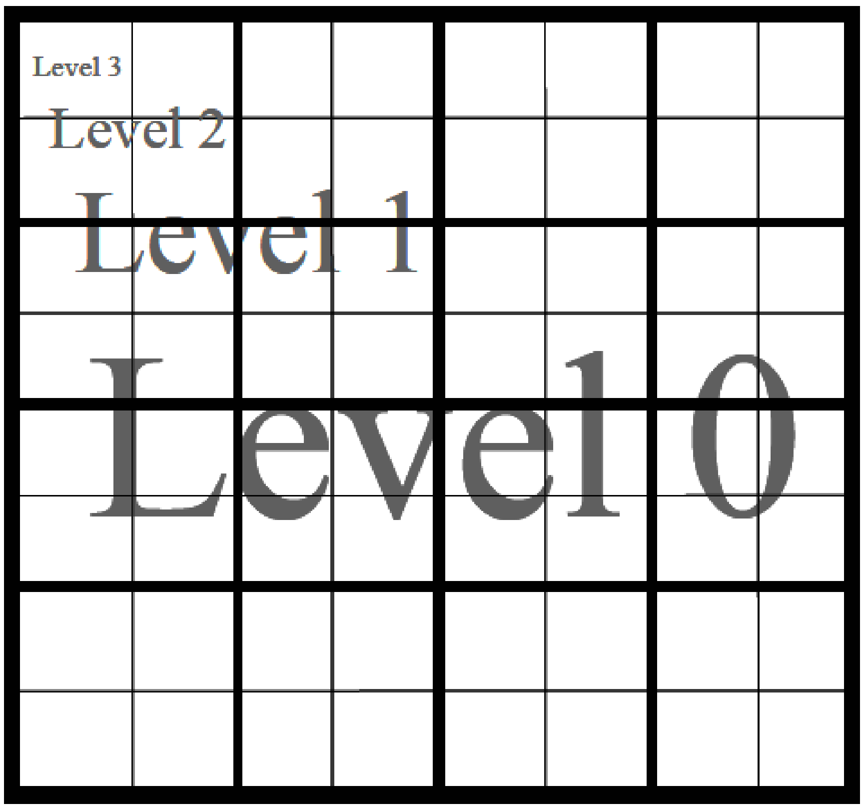
- Approaches include:
  - Data preprocessing
    - Sorting
    - Building lists (such as neighbor lists): requires memory, potentially can be avoided;
- Operations with data during the FMM algorithm execution:
  - Operations on data sets;
  - Search procedures.

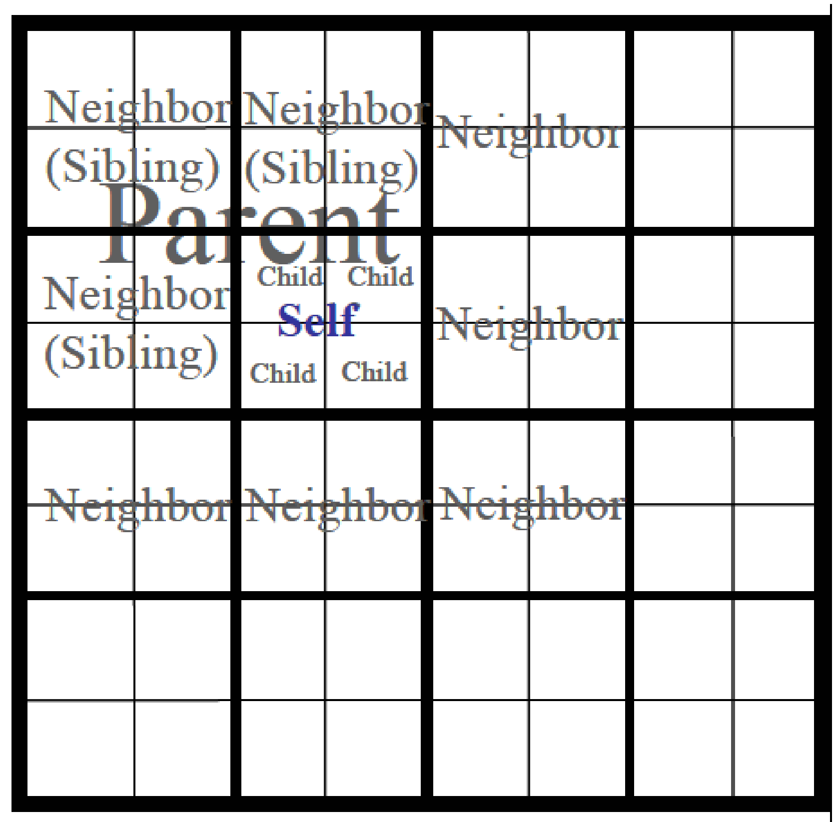
FMM Data Structures contd..

- Preferable algorithms:
  - Avoid unnecessary memory usage;
  - Use fast (constant and logarithmic) search procedures;
  - Employ bitwise operations;
  - Can be parallelized.

- We will consider a concept of 2^d-tree.

Hierarchy in 2^d-tree





Hierarchy in 2^d-tree

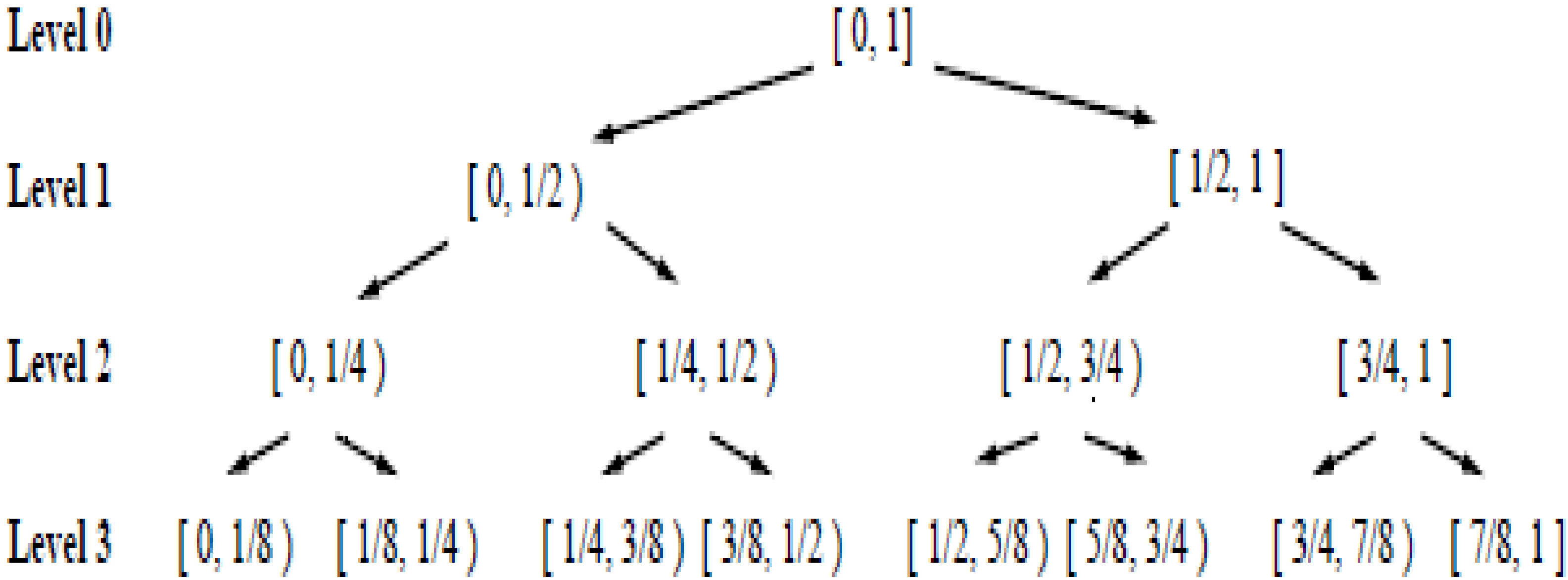
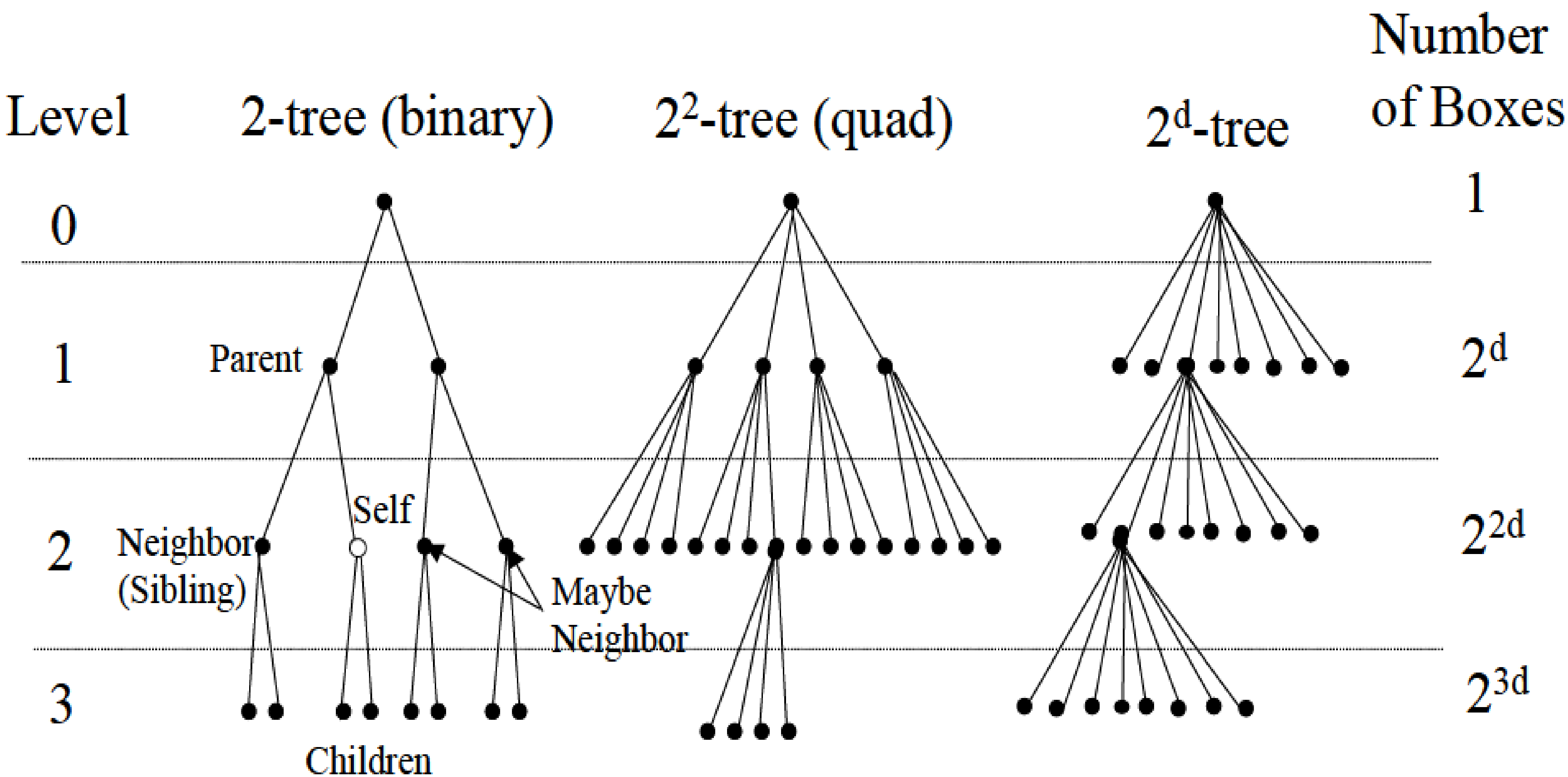
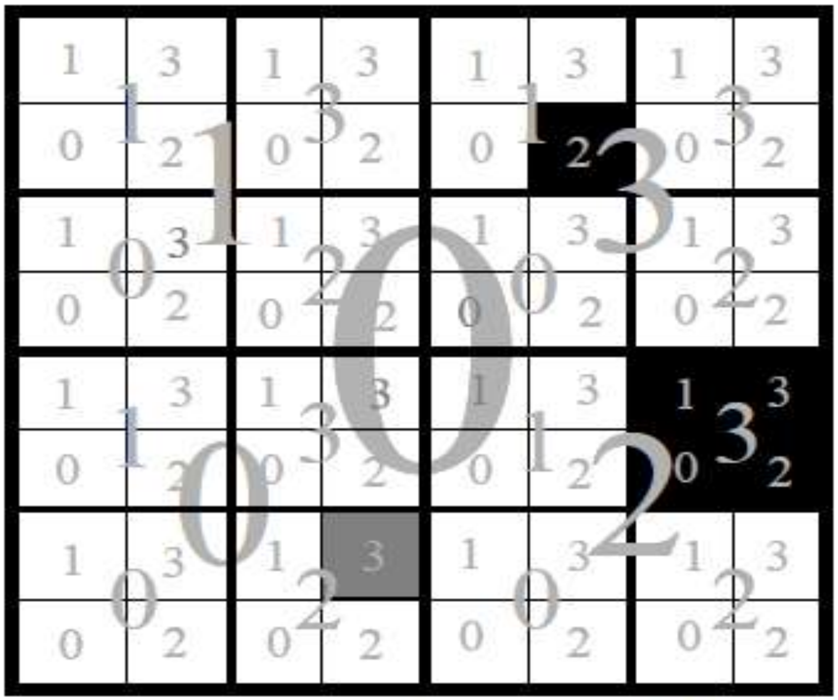


FIG. 2. Binary tree structure induced by a uniform subdivision of the unit interval.

2^d-trees


Level
2-tree (binary)
2^2-tree (quad)
2^d-tree
Number of Boxes
0
1
2
3
Parent
Self
Neighbor
(Sibling)
Maybe
Neighbor
Children
1
2^d
2^{2d}
2^{3d}

Hierarchical Indexing in 2^d-trees. Index at the Level.



Indexing in quad-tree

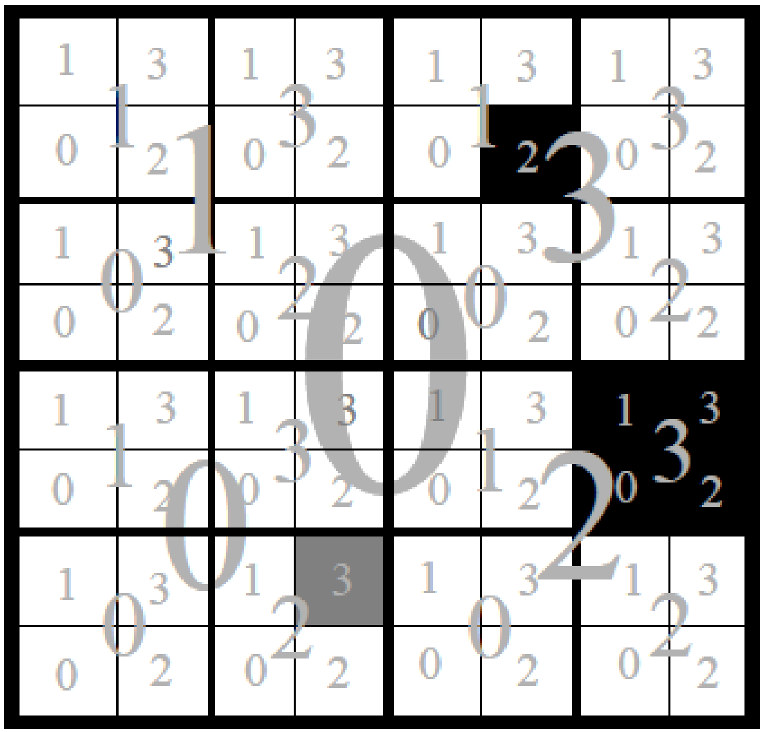
The large black box has the indexing string (2,3). So its index is $23_4=11_{10}$.

The small black box has the indexing string (3,1,2). So its index is $312_4=54_{10}$.

In general: Index (Number) at level l is:

$$Number = \left(2^d\right)^{l-1} \bullet N_1 + \left(2^d\right)^{l-2} \bullet N_2 + \ldots + 2^d \bullet N_{l-1} + N_l.$$

Universal Index (Number)



The large black box has the indexing string (2,3). So its index is $23_4=11_{10}$ at level 2

The small gray box has the indexing string (0,2,3). So its index is $23_4=11_{10}$ at level 3.

In general: Universal index is a pair:

$$UniversalNumber = (Number, l)$$

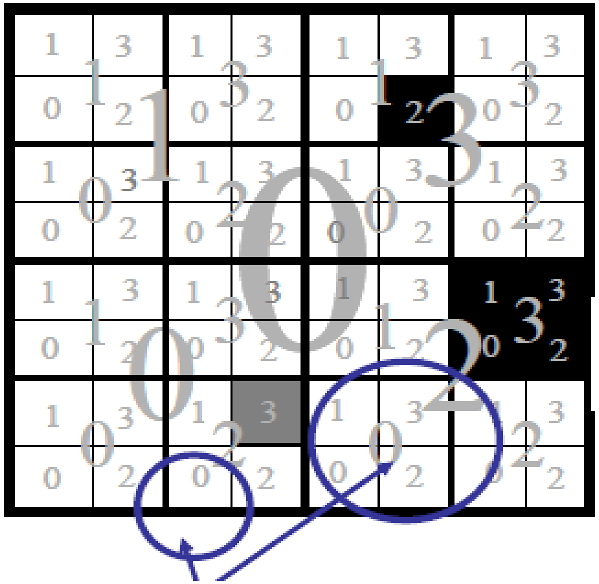
Parent Index

Parent's indexing string:

$$Parent(N_1, N_2, \dots, N_{l-1}, N_l) = (N_1, N_2, \dots, N_{l-1}).$$

Parent's index:

$$Parent(Number) = \left(2^d\right)^{l-2} \bullet N_1 + \left(2^d\right)^{l-3} \bullet N_2 + \dots + N_{l-1}.$$



Parent index does not depend on the level of the box! E.g. in the quad-tree at any level

$$Parent(11_{10}) = Parent(23_4) = 2_4 = 2_{10}.$$

Parent's universal index:

$$Parent((Number, l)) = (Parent(Number), l-1).$$

Algorithm to find the parent number:

$$Parent(Number) = \left[Number/2^d\right]$$

For a box #23 (gray and black) the parent box index is 2

Children Indexes

- Children indexing strings:
  - $Children(N_1,N_2,..,N_{l-1},N_1) = \{(N_1,N_2,..,N_{l-1},N_{1,}\ N_{l+1})\}$, $N_{l+1} = 0,\ldots,2^d\text{-}1$.
- Children universal indexes:
  - $Children((Number, l) – (Children(Number), l + 1)$
- Algorithm to find the children numbers:
  - $Children(Number) = \{ 2^d * Number + j\}, j = 0,\ldots,2^d\text{-}1$

Examples

- Problem: Using the above numbering system and decimal numbers find <u>parent</u> box number for box # 5981 in oct-tree.
  - *Find the integer part of division of this number by 8 [5981/8] = 747*
- Using the above numbering system and decimal numbers find <u>children</u> box number for box #100 in oct-tree.
  - *Multiply this number by 8 and add numbers from 0 to 7 -> (800, 801, 802,…….. 807).*

Neighbor finding

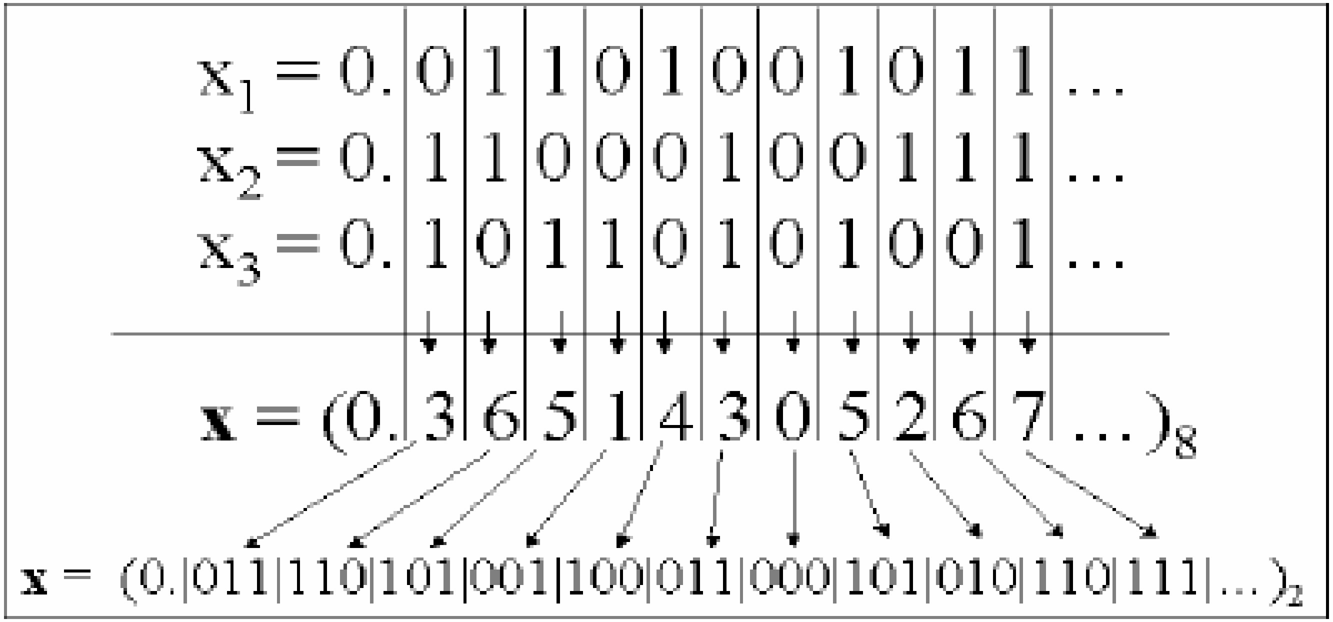
  – *Neighbor((Number, level))=Number±1*

- If the neighbor number at level l equal 2^l or -1 we drop this box from the neighbor list.

- Find all neighbors of box #31 (decimal) at level 5 of the binary tree.
  – *The neighbors should have numbers 31-1 = 30 and 31 +1 =32. However, 32 =2^5, which exceeds the number allowed for this level. Thus, only box #30 is the neighbor*

Bit Interleaving

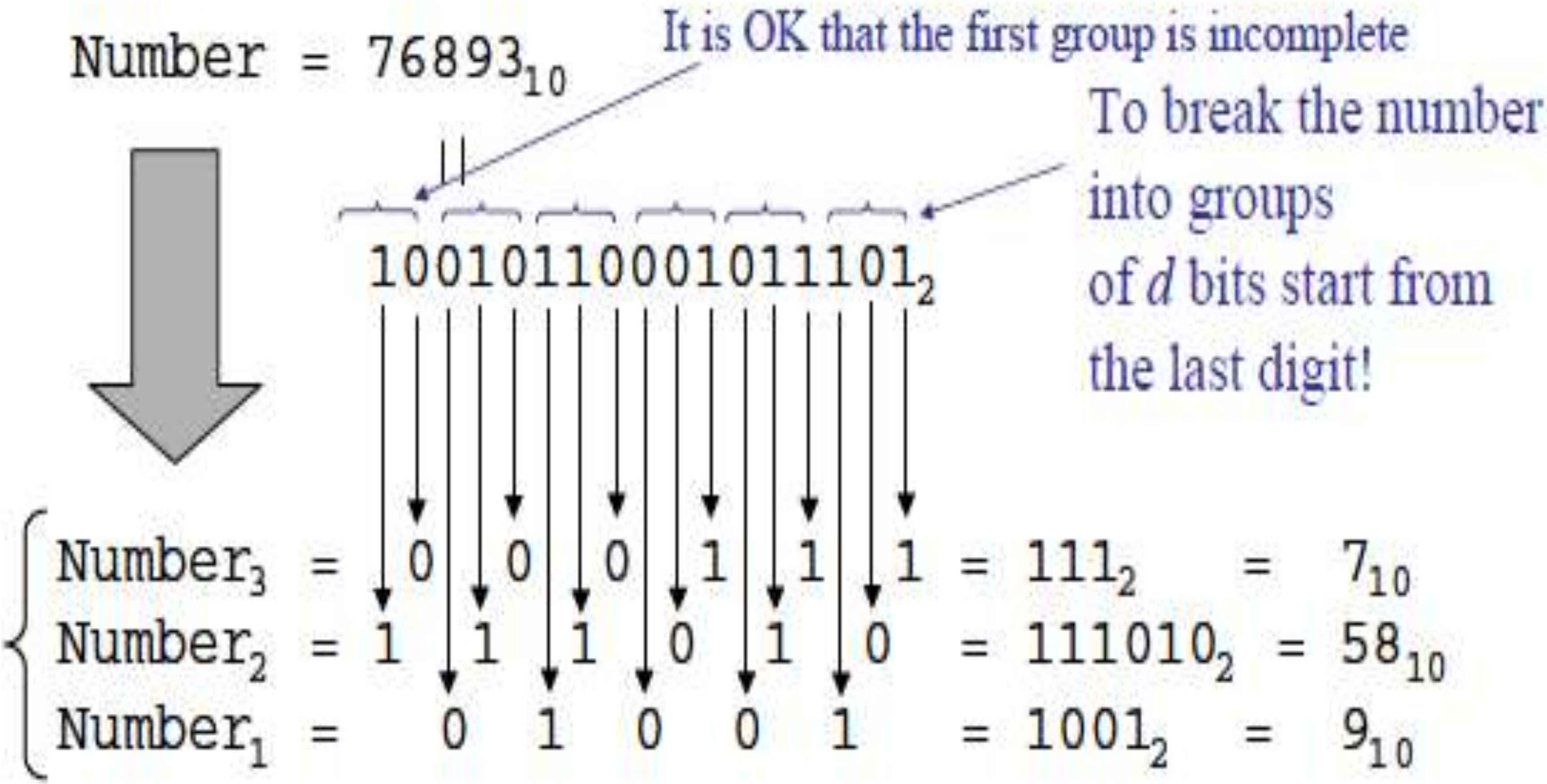
- It is represented as
  - $X = (0.b_{11}b_{21}..b_{d1}b_{12}b_{22}..b_{d2}...b_{1j}b_{2j}..b_{dj}..)_2$
- It can also be represented as
  - $X= (0.N_1N_2N_3...N_J...)_{2^d}$, $N_j = (b_{1j},b_{2j}.... b_{dj})_2$, $j=1,2,....$, $Nj = 0,....2^d -1$



Bit Deinterleaving

- Convert the box number at level l into binary form
  - Number = $(b_{11}b_{21}..b_{dl}b_{12}b_{22}...b_{d2}..b_{l1}b_{l2}..b_{dl})_2$
- Then we decompose this number into numbers that will represent d coordinates:
  - Number1 = $(b_{11}b_{12}...b_{l1})_2$
  - Number 2 =$(b_{21}b_{22}...b_{l2})_2$
  - Number3 = $(b_{dl}b_{d2}..b_{dl})_2$

Bit Deinterleaving contd..

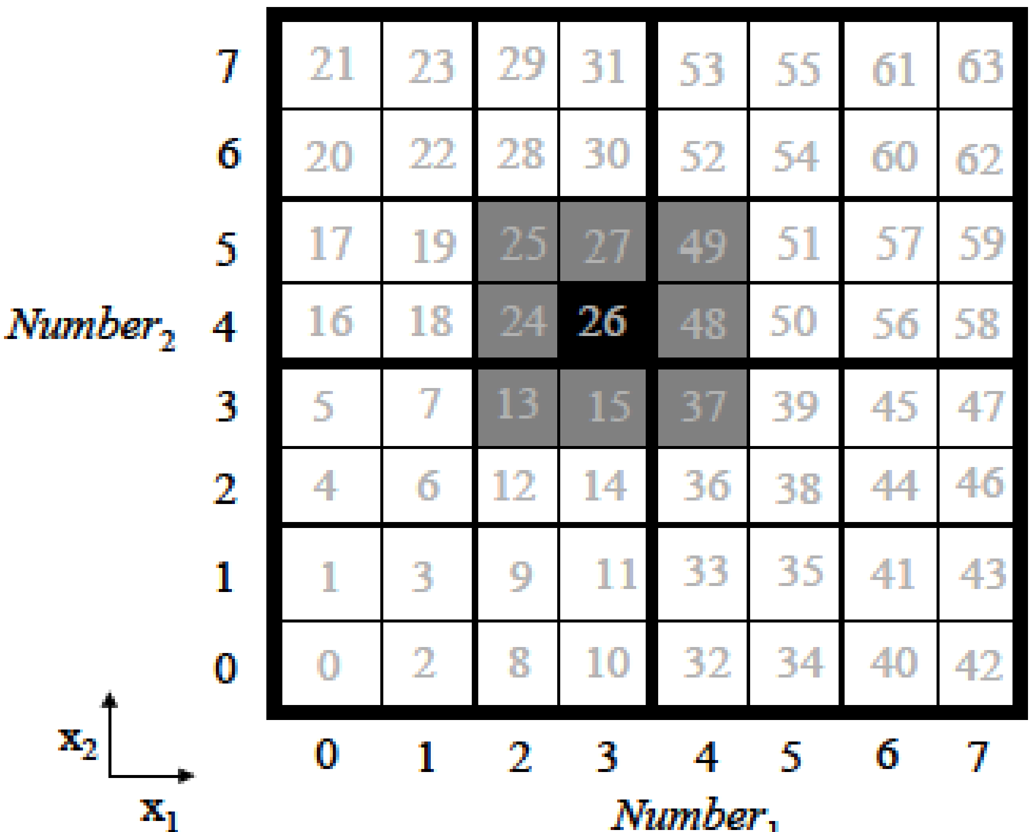


Neighbor Finding

- Step 1: Deinterleaving:
  - $Nos \rightarrow \{ Nos_1 \ldots\ldots Nos_d \}$
- Step 2: shift of the coordinate numbers
  - $Nos^{+}_{k} = Nos_k + 1,\ Nos^{-}_{k} = Nos_k - 1,\ \ k = 1 \ldots d$
- $Sk = \{ Nos^{-}_{k}, Nos_k\ Nos^{+}_{k} \}, \qquad Nos_k \neq 0,\ 2^l - 1$
- $\{ Nos^{-}_{k}, Nos_k\ Nos^{+}_{k} \}, \qquad Nos_k = 0,\ \ k = 1 \ldots d$

  $\{ Nos^{-}_{k}, Nos_k\ Nos^{+}_{k} \}, \qquad Nos_k =,\ 2^l - 1$
- *Step 3: Convert it into binary and apply interleaving*

Example of Neighbor Finding



$26_{10} = 11010_2$

deinterleaving

$(11,100)_2 = (3,4)_{10}$

generation of neighbors

(2,3) ,(2,4),(2,5),(3,3),
(3,5),(4,3),(4,4),(4,5)
=
(10,11),(10,100),(10,101),
(11,11),(11,101),(100,11),
(100,100),(100,101)

interleaving

1101,11000,11001,1111,11011,100101,110000,110001
= 13, 24 ,25, 15, 27, 37, 48, 49

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