IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 9, November 2014.
www.ijiset.com
ISSN 2348 – 7968

Parallel Search Technique Using Quad Based Partitioning Mechanism for Dataset

Ajay Jain[1], Dr. Neelendra Badal[2]

[1]Computer Science and Engineering Department, Kamla Nehru Institute of Technology, Sultanpur, UP 228118, India
[2]Computer Science and Engineering Department, Kamla Nehru Institute of Technology, Sultanpur, UP 228118, India

Abstract

Quadtree Block Partition is the technique through which the matrix can be partitioned into small pieces or blocks. By partition the matrix, it is very easy to search an item in the matrix in parallel. If the matrix is partitioned into discrete parts, then the search technique can be applied on those partitions in parallel. The use of quadtree partition in this paper is to partition the matrix into small blocks or pieces for searching the elements in matrix in parallel. In this paper a new algorithm is proposed for parallel searching by converting the dataset into matrix and form that matrix into quadtree through which parallel search technique can be applied on those quad blocks and can search the item in all quads concurrently.

Keywords: *Parallel Computing, Parallel Processing, Parallel Searching, Quadtree Decomposition, Quadtree Representation of Matrices, Metadata Management.*

1. Introduction

In this paper, the proposed quadtree matrix formation scheme is being described, quadtree concept for partition is going to discuss for decomposition of the matrix block. The matrix will be generated from the primary key attribute of the database table. Further recursively decomposition of the matrix is done till certain level and through this scheme the dataset will be converted into proposed quadtree matrix blocks so that an item can be easily searched from the dataset.

This paper is also going to deal with parallel processing. Parallel searching concept is used in this paper. For parallel searching, firstly partitions the dataset matrix into quadtree and finds that which quad block has the searching item by using some previously defined constraints or metadata information.

For partition the matrix into quadtree, there is a need of metadata management. The matrix can be decomposed into quadtree matrix easily by using metadata information. Metadata information can be made by forming the database into matrix and by placing the related data to the appropriate block. There are many constraints which are used in database for access the database easily. Using these constraints the quadtree partition can be applied on that database and easily access the required data in the database. Quadtree partition technique partitions the matrix into four matrices at one level simultaneously.

Parallel search is very fast as compared to sequential search. Parallel search is very useful for doing the work concurrently and easily. Parallel searching is also useful for multiple resources utilization at a time and for getting better performance. Therefore a new parallel search technique introduced here using quadtree block partition.

2. Literature Review

A quadtree [1] is a tree data structure in which each internal node has exactly four children. Quadtree is most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular, or may have arbitrary shapes. This data structure was named a quadtree by Raphael Finkel and J.L. Bentley in 1974. A similar partitioning is also known as a Q-tree.

Symbolic and Algebraic Computation, Lecture Notes in Computer Science Volume 358, 1989, S. Kamal Abdali and David S. Wise did experiments with quadtree representation of matrices. S. Kamal Abdali and David S. Wise states in [2], the quadtree matrix representation has been recently proposed as an alternative to the conventional linear storage of matrices. If all elements of a matrix are zero, then the matrix is represented by an empty tree, otherwise it is represented by a tree consisting of four sub trees, each representing, recursively, a quadrant of the matrix. Using four-way block decomposition, algorithms on quadtree accelerate on blocks entirely of zeroes, and thereby offer improved performance on sparse matrices. In this work the results of experiments done with a quadtree matrix [9] package implemented in reduce to compare the performance of quadtree representation with reduce's built-in sequential representation of matrices. Tests on addition, multiplication, and inversion of dense, triangular, tri-diagonal, and diagonal matrices both symbolic and

ISSN 2348 – 7968

numeric of sizes up to 100×100 show that the quadtree algorithms perform well in a broad range of circumstances, sometimes running orders of magnitude faster than their sequential counterparts.

Another attractive attribute of the quadtree representation of matrices is that it unifies computation on both dense and sparse matrices. That is, this single representation can represent both dense and non-dense matrices with relatively efficient use of space, and this single family of algorithms manipulates both sparse and non-sparse matrices with relatively conservative use of time [3]. There are better specialized structures and algorithms for extremely dense or extremely sparse matrices, but no other approach avoids a dichotomy of performance across the spectrum. Although not often a design criterion in computer algebra systems, efficient handling of both sparse and non-sparse problems is a welcome dividend of this approach [4].

The earliest applications of parallel processing [11] have been in the area of matrix computations. For conventional matrices, the parallelism [12] is gained by array processing in which the number of processors needed is comparable to the matrix size. In contrast, the parallelism quadtree matrix computations arise most naturally from the for-way recursive decomposition of the tree. It thus seems the quadtree matrices can offer parallel computing [7] opportunity even in the environments where the number of processors is small and fixed.

3. Quadtree Matrix Decomposition

In this section, the matrix decomposition of the dataset using quadtree partition technique is described. Using quadtree partition technique, the matrix block is decomposed into four partitions of quadtree matrices and those matrices are also decomposed into other quadtree matrices and so on. This process will continue recursively till certain level. In this process of quadtree decomposition [5], the matrix of size *n* x *n* is decomposed into four matrices of size *n/2* x *n/2* and these matrices will further decomposed into matrices of size *n/4* x *n/4* and so on.

For decomposing the matrix into quadtree matrix, firstly the dataset is converted into square matrix of *n* x *n* by adding some null elements. It is noticed that for splitting matrix into quadtree matrix, n must be even number.

By using given formula, the value of maximum level (L) of the matrix can be calculated up to which the matrix can be decomposed.

For matrix of size *n* x *n*, L= Max(x) such that 2^x *divides* *n* where x= 1, 2, 3, 4…….

Now the decomposition of the matrix is being started. There are two types of decomposition used in this paper.

3.1 Partially Split

Matrix can be decomposed partially into the quadtree matrix. It means, end user can decide the level of matrix decomposition which can be from 1 to L where L is the maximum level of matrix decomposition. It is asked from the user that he wants to partially split the matrix or fully split the matrix. If he wants to partially split the matrix, then user enters the level of the matrix up to which matrix will be decomposed.

3.2 Fully Split

Matrix can be decomposed fully into the quadtree matrix. In this case matrix will be decomposed up to the maximum decomposition level. In partially split, if user enters the value of level greater than or equal to maximum level L then in that case, matrix will also be decomposed up to the maximum level L.

4. Description Of Quadtree Matrix Formation

In this section, it is discussed how to split the matrix in well-defined manner so that an item can be easily searched and database can also be maintained into organized manner. The matrix is decomposed in quadtree form using given formula so that the matrix partition can be formed in organized manner. Through this formula the matrix can be easily decomposed and store the matrices at proper location. There is a control point in this formula which shows the level of the matrix partition block and also has relevant information to its parent matrix block. Using this information it is very easy to regenerate the initial matrix by combining matrices partition for finding the exact location of the searching item in the initial matrix. This formula is very useful in finding the required information from the database by converting the database in to metadata using some constraints. The formula is as follows:

$$CPn_{(i,\,j)} = \begin{matrix} CPn{+}1_{(2i-1,\,2j-1),} \\ CPn{+}1_{(2i-1,\,2j),} \\ CPn{+}1_{(2i,\,2j-1),} \\ CPn{+}1_{(2i,\,2j),} \end{matrix}$$

Fig 1: Quadtree Partition Formula

n= 0, 1, 2, 3, 4... …L (Level)

i= 1, 2, 4…… 2^n

j= 1, 2, 4…… 2^n

ISSN 2348 – 7968

The quadtree matrix formation using this formula is shown in fig 2. In this figure, CP is the control point, and it is the initial matrix block at level 0. This matrix is to be partitioned into quadtree. Therefore, the matrix is decomposed into four sub matrices. The label of the sub matrices is generated using given formula [$CP1_{11}$, $CP1_{12}$, $CP1_{21}$, $CP1_{22}$]. $CP1_{11}$ indicates that the level of the decomposition is 1, and matrix is the block or the sub matrix that is in the position of 1st row and 1st column block of the initial matrix or super matrix.

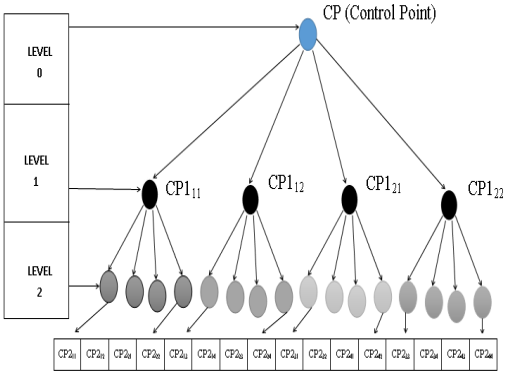


Fig 2: Quadtree Formation

There is *8x8* matrix block in fig 3 which has total 64 elements. Because n=8 is an even number therefore quadtree decomposition can be done of that matrix. The maximum level L can be calculated for that matrix.

8 can be divided by 2^1, 2^2, 2^3 (x= 1, 2, 3)

L= Max(x) = 3 (The maximum level of the decomposition is 3).

It means, the *8* x *8* matrix will be decomposed into four *4* x *4* sub matrices at first level, sixteen *2* x *2* sub matrices at second level and sixty four *1* x *1* sub matrices or single cell elements at third level as shown in fig 4, fig 5 and fig 6 respectively.

MATRIX [CP]: (LEVEL 0)

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Fig 3: n x n Initial Matrix Block

LEVEL 1

$CP1_{11}$

1	2	3	4
9	10	11	12
17	18	19	20
25	26	27	28

$CP1_{12}$

5	6	7	8
13	14	15	16
21	22	23	24
29	30	31	32

$CP1_{21}$

33	34	35	36
41	42	43	44
49	50	51	52
57	58	59	60

$CP1_{22}$

37	38	39	40
45	46	47	48
53	54	55	56
61	62	63	64

Fig 4: n/2 x n/2 Matrix Block

LEVEL 2

$CP2_{11}$

1	2
9	10

$CP2_{12}$

3	4
11	12

$CP2_{13}$

5	6
13	14

$CP2_{14}$

7	8
15	16

$CP2_{21}$

17	18
25	26

$CP2_{22}$

19	20
27	28

$CP2_{23}$

21	22
29	30

$CP2_{24}$

23	24
31	32

$CP2_{31}$

33	34
41	42

$CP2_{32}$

35	36
43	44

$CP2_{33}$

37	38
45	46

$CP2_{34}$

39	40
47	48

$CP2_{41}$

49	50
57	58

$CP2_{42}$

51	52
59	60

$CP2_{43}$

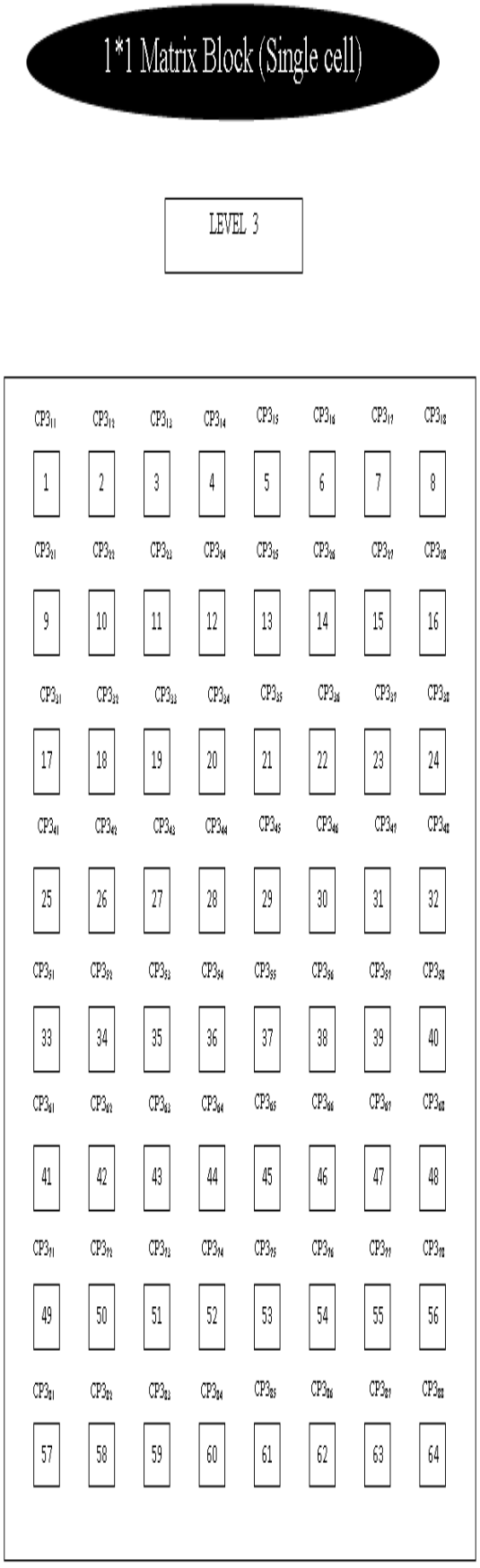
53	54
61	62

$CP2_{44}$

55	56
63	64

Fig 5: n/4 x n/4 Matrix Block

Fig 6: n/8 x n/8 Matrix Block



5. Parameters

There are some kind of parameters of quadtree partition is as follows.

5.1 Sub Quad Block:

The number of sub quad block indicates how many blocks will be generated at n^{th} level from the initial matrix block. The number of sub quad block can be calculated by given formula.

$$\text{No. of Sub Quad Block } N_{SQB} = 4^n * N_{QB}$$

5.2 Length of Sub Quad Block:

The length of the sub quad block indicates the size of the sub quad matrix block which will be generated at nth level from the initial matrix block.

$$\text{Length of Sub Quad Block } L = \frac{1}{2^n} * \text{length of QB}$$

6. Description of Parallel Searching Technique:

In this section, the implementation of a parallel searching [6] algorithm is represented, which is used for finding the required information in quadtree matrices which is generated in previous section from the dataset matrix. The purpose of this proposal is to reduce the searching time in a multicore [10] machine with shared memory. The proposed algorithm implementation and the performance tests are done in Matlab Parallel Toolbox. In this process of parallel searching, each quad is assign to a different core processor and applies searching on that quad as shown in below diagram.

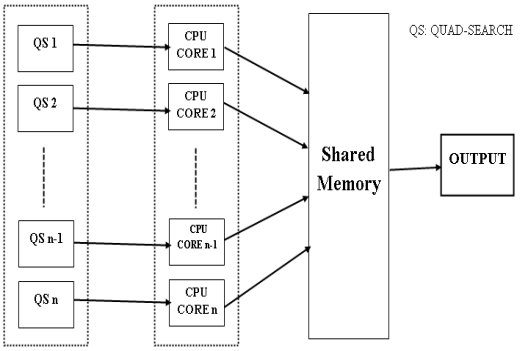


Fig 7: Parallel Searching in Quad Blocks

ISSN 2348 – 7968

There are some of the steps of parallel searching algorithm. These steps are used in this algorithm for applying searching technique on the quad based matrices in parallel.

Step 1- Enter the Searching Item:
In the first step of parallel searching algorithm, there is a need of the value of the item that is to be searched. There are two ways of enter the searching item.

- *By using Command Window:*

In this process, it is asked to the user at command window for entering the searching item, and program will wait till user enters the searching item. After entering the searching item by user, further processing of the program is started.

- *By using Array list:*

Array list can be previously generated by the user for searching many items one by one. The searching item is taken from the array list one by one and the searching is done on that item accordingly.

Step 2- Analysis of the Quad blocks:
In second step, the analysis of the quad blocks is done as shown in fig 8. There are four Quad blocks available at first level which is generated from the initial matrix. Using some constraints it is decided which quad block may have the searching item. After deciding which quad has the searching item at first level, the second level of the quadtree is used for further deciding the appropriate quad block of the searching item and now the searching of the item is done at sub quadtree of that block. If there is any other constraint at second level then that constraint is used to decide which quad has the searching item and goes to third level and so on. At which level the block of searching item could not be decided, the parallel search technique [8] is applied on those four blocks using four-core processors in parallel.

In this algorithm a constraint is used at first level. There are some of the steps for applying constraint at first level in the case of *n* x *n* size matrix block.

- If the value of searching item is less than *n* x *n/2* then item may exist in first or second matrix block.
- If the remainder of item when the value of item is divided by n, is in the range of 1 to *n/2* then item must be available in first Quad block otherwise if the remainder is 0 or greater than *n/2* then item may be available in second Quad block.
- Otherwise, if the item is greater than *n* x *n/2* and less than *n* x *n* then item may exist in third or fourth matrix block.
- If the remainder of item when the value of item is divided by n, is in the range of 1 to *n/2* then item may be available in third Quad block otherwise if the remainder is 0 or greater than *n/2* then item may be available in fourth Quad block.
- If the item is greater than *n* x *n* then item must be out of range.

For example if there is *8* x *8* matrix block and user enters the value of searching item is 38 then the above procedure is applied to find the relevant Quad for searching the item.

1. Because size of the matrix is *8* x *8* so the value of n is equal to 8 and the value of *n* x *n/2 = 8 * 8 / 2 = 32*, but value of item is greater than this value so the item is not exist in first or second matrix block.
2. The value of *n* x *n* = *8* x *8* = 64, and the value of item is greater than 32 and less than 64 then the item may exist in third or fourth matrix block.
3. Now find the remainder when the value of item is divided by n.
4. Rem(item, n) = Rem(38, 8) = 6, but it is not in the range of *1 to 4 (n/2 = 4)* so the item is not available in third Quad block.
5. Now check the condition for fourth Quad block. The remainder is not 0 but it is greater than 4 (n/2) so the item must be available in fourth Quad block.

This way the relevant Quad block of the searching item can be found. After finding the relevant quad block, all the other quad blocks except relevant quad block is discarded and apply further searching on the sub quad tree of that quad block.

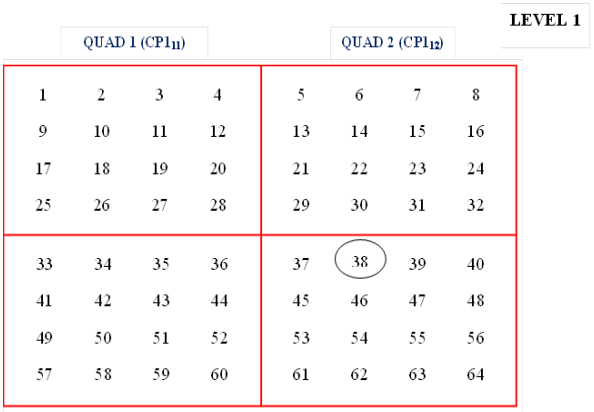


Fig 8: Quad Block Analysis

Step 3- Allotment of Processors to the Quads:
In this step, allotment of the processors to each quad is done as shown in fig. 9. There are four core processors for allotment. After getting the relevant quad block from the 2nd step, one core processor is allotted to each sub quad of the relevant Quad block for further processing of the searching element.
As from the above analysis, it is found that the searching item is available in fourth quad, it means the 4th quad block is the relevant quad block for searching the item. So all other quad blocks except 4th

ISSN 2348 – 7968

quad block is discarded at first level. And apply searching on the sub quads of the fourth quad blocks by allotting a core processor to each quad block.

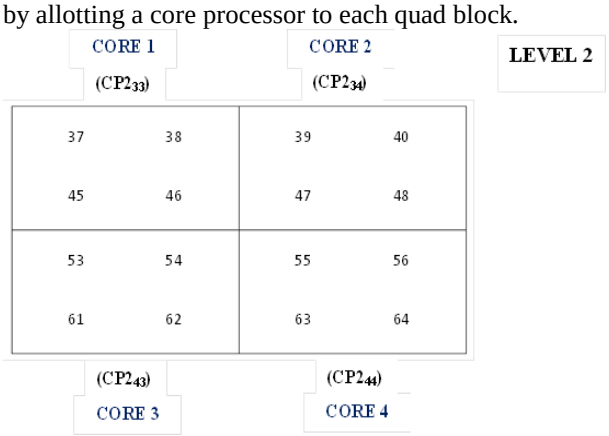


Fig 9: Allotment of core processor

Step 4- Perform Searching on each Quad:
After allotment of the processor, parallel searching is done using four core processors. Each core searches for the item in its quad using any serial search technique. Only one quad has the searching item, and other quads do not have the item. Because the matrix is made from primary key attribute therefore it could not have duplicate value. Which quad has the searching item, that gives in output its position in the quadtree and index of the item in that quad. All other quads give the output that 'item is not found'.

Step 5- Position of the Item:
After searching has been completed, there is a need to get the exact position of the item in the appropriate quad matrix and its place in quadtree. In above example, the item is 38 which is found in $CP2_{33}$ quad block at second level. And index of the item is (1, 2). $CP2_{33}$ is the address of the position of the quad block in quadtree. It means the quad is available at second level in 3rd row and 3rd column, through which the exact position of the matrix can be found. In above example item found in 4th block at first level and further the item is found in 1st block at second level.

Step 6- Searching of another Item:
There are two ways of searching of another item.

- *By using Command Window:*

After completing the searching of the previous item, it is asked to the user if he wants to search another item otherwise terminate the program. If he selects the option of searching again then it is asked to the user at command window for entering another searching item, and program will wait till user enters the searching item. After entering the searching item by user, further processing of the algorithm is started and all the above steps will follow again for searching another item. This process will continue until user terminates the program

- *By using Array list:*

If the previous searching is done using array list then algorithm automatically takes another element from the array list. This process will continue until last element of the array list is not processed. When all searching element of the array list will be processed, the program will terminate automatically.

7. Performance:

An experiment has been carried out to evaluate performance of proposed parallel search methodology using quadtree partition technique. It has been carried out on a system with core2Duo processor, 4GB RAM, Matlab toolbox and MS Access database. The performance of the proposed Parallel search technique on database system is compared with performance of sequential search at 1 core processor. In the proposed model, performance evaluation is done by neglecting the cost of communication between the core processors.

7.1 Performance on Static Size Matrix Block When Number Of Searching Element Increases

The performance of the sequential search and parallel search is compared when the size of the matrix block is static and the number of searching element increases at single core processor.

Table 1: Performance on Block Size 100 x 100

S. No.	No. of searching Item	Sequential Time (in seconds)	Parallel Time (in seconds)
1.	1	0.018807	0.014975
2.	10	0.121198	0.090542
3.	100	1.111096	0.981576
4.	1000	11.515309	10.393090
5.	10000	81.187911	67.186979

ISSN 2348 – 7968

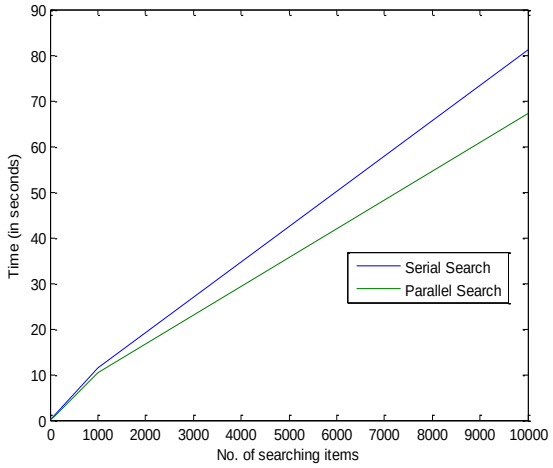


Fig 10: Graph between Time and No. of Searching Item at block 100 x 100

Table 2: Performance on Block Size 1000 x 1000

S. No.	No. of searching Item	Sequential Time (in seconds)	Parallel Time (in seconds)
1.	1	0.859120	0.334872
2.	10	8.724550	2.301308
3.	100	87.142637	22.208905
4.	1000	1067.137784	266.166367
5.	10000	6736.023786	1797.819639

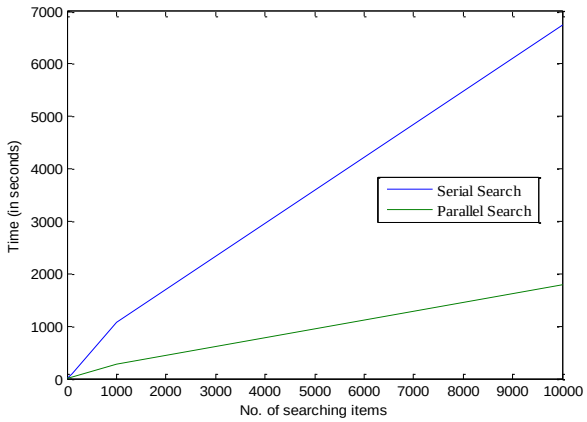


Fig 11: Graph between Time and No. of Searching Item at Block 1000 x 1000

7.2 Performance on Dynamic Size Matrix Block When Number Of Searching Item Is Fixed

In this section, the performance of parallel search is compared with sequential search when number of searching element is fixed and size of the matrix block varies. The algorithm is applied at matrix blocks of different sizes for searching 100 elements simultaneously and finds the computing time in searching 100 elements at each matrix block. When the graph of the computing time of parallel search and sequential search is plotted, it can be seen in the below graph that the time difference increases rapidly. It means when the size of the matrix block increases, the parallel algorithm runs much faster than sequential search algorithm.

Table 3: Performance on 100 searching element

S. No.	Matrix Block Size (n x n)	Sequential Time (in seconds)	Parallel Time (in seconds)
1.	100	1.132498	1.040419
2.	200	3.766016	1.686504
3.	300	8.162527	2.754849
4.	400	14.129236	4.254465
5.	500	22.424957	6.250672
6.	600	32.303895	8.340900
7.	700	42.069709	11.546755
8.	800	55.155839	14.453790
9.	900	69.023648	17.956568
10.	1000	85.448375	22.312300

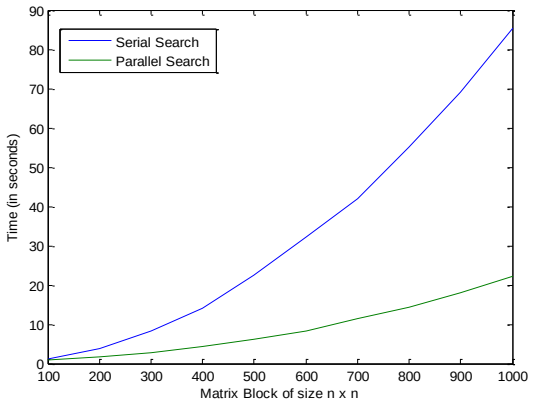


Figure 12: Graph between Time and n x n Matrix Block for 100 searching item

8. Conclusion And Future Scope

In this paper the quadtree matrix formation of dataset is done, the decomposition of the matrix into quadtree is implemented and the new technique for parallel

ISSN 2348 – 7968

searching using quad based partition technique is implemented. This paper concludes that the proposed parallel search technique is much better and very useful in case of very large size databases for searching large number of items simultaneously.

The future works for the proposed algorithm are:

- Meta data management may be incorporated in better way.
- Different applications may be implemented.
- Core management may be introduced in future.
- Distribution policy may be incorporated.
- An algorithm may be used that would constantly check performance of parallel searching at each level of quadtree decomposition

9. References

[1] Gisli R. Hjaltason, Hanan Samet, "Speeding up construction of PMR quadtree-based spatial indexes" *in Computer Science Department, Center for Automation Research, and Institute for Advanced Computer Studies, University of Maryland, College Park, USA*, March 25, 2002, pp. 112-116.

[2] S. Kamal Abdali, David S. Wise., "Experiments with Quadtree representation of matrices", *Symbolic and Algebric Computation, International Symposium ISSAC 88, Rome, Italy*, July 1988, pp. 96-100.

[3] D.S. Wise & J. Franco, "Costs of quadtree representation of non-dense matrices", *Technical Report No. 229, Computer Science Department, Indiana University* (October, 1987).

[4] D. S. Wise., "Parallel decomposition of matrix inversion using quadtrees", *in Hwang, K., Jocobs, S.J., and Swartzlander E. E. (Eds.), Proc. 1986 International Conference On Parallel Processing, IEEE Computer Society Press, Washington*, 1986, pp. 92-99.

[5] C. Faloutsos, H.V. Jagadish and Y. Manolopoulos, "Analysis of the n. dimensional quadtree decomposition for arbitrary hyperectangles", *Proc. TKDE*, 9, no. 3 (1997), pp. 373-383.

[6] Ted Ralphs, Laszlo Ladanyi, "Implementing Scalable Parallel Search Algorithms for Data-Intensive Applications" *in International Conference on Computational Science 2002*, Tuesday April 23, 2002, pp. 4-17.

[7] Asanovic, Krste et al., "The Landscape of Parallel Computing Research: A View from Berkeley", *University of California, Berkeley. Technical Report No. UCB/EECS-2006-183.*, December 18, 2006., pp. 12-15

[8] Danny Z. Chen., "Efficient parallel binary search on sorted arrays", *Technical Report 1009, Purdue University, Department of Computer Science*, August 1990.

[9] David S. Wise, "Matrix algorithms using quadtrees", *in Proc. ATABLE- 92*, 1992, pp. 11-26.

[10] Nathalie Revol, Philippe Theveny, "Parallel Implementation of Interval Matrix Multiplication" *in University de Lyon LIP (UMR 5668 CNRS - ENS Lyon - UCB Lyon 1 - INRIA), ENS de Lyon, France*, March 2013, pp. 6-9.

[11] Lin C. and Snyder L., "Principles of Parallel Programming", *Parallel Processing via MPI & OpenMP, M. Firuziaan, O. Nommensen. Linux Enterprise*, 10/2002.

[12] Bora Ucar, "Parallel Sparse Matrix-Vector Multiplies And Iterative Solvers", *in the institute of engineering and science of Bilkent University*, August, 2005, pp. 27-31.

Ajay Jain is the student of Master of Technology in Department of Computer Science & Engineering at Kamla Nehru Institute of Technology (KNIT), Sultanpur, India. He has received his Bachelor of Technology degree in 2012 from Radha Govind Engineering College (RGEC), Meerut, India in Information of Technology.

Dr. Neelendra Badal is an Assistant Professor in the Department of Computer Science & Engineering at Kamla Nehru Institute of Technology (KNIT), Sultanpur (U.P), India. He received B.E. (1997) from Bundelkhand Institute of Technology (BIET), Jhansi in Computer Science & Engineering, M.E. (2001) in Communication, Control and Networking from Madhav Institute of Technology and Science (MITS), Gwalior and PhD (2009) in Computer Science & Engineering from Motilal Nehru National Institute of Technology (MNNIT), Allahabad. He is Chartered Engineer (CE) from Institution of Engineers (IE), India. He is a Life Member of IE, IETE, ISTE and CSI-India. He has published about 30 papers in International/National Journals, conferences and seminars. His research interests are Distributed System, Parallel Processing, GIS, Data Warehouse & Data mining, Software engineering and Networking.