Design a pathway/genome expert system using a prolog machine incorporated with a parallel hardware searcher§

Wenlung Shu[1] and Junming Lan[2]
Bioinformatics Department[1] and CSIE Department[2]
Chung Hua University, Hsin Chu, Taiwan
wlshu@chu.edu.tw

Abstract

It is well recognized that processing complex pathway/genome databases (PGDB) is a very time consuming task. A specific prolog machine which is incorporated with a parallel hardware searcher is proposed to process PGDB in this paper. These databases and the relevant programs are represented as prolog horn clauses (facts and rules) which are treated as objects, and stored in DRAM to increase access time. Search module is a parallel hardware searcher with 9 processor units and 9 corresponding DRAM modules. Index files of object databases can be efficiently processed in search module, and each record of index file contains a search value of an object and corresponding physical address of this object.

Java is used to integrate search module and SWI prolog interpreter. Search module finds all goal related objects by searching index files of object databases continuously. Therefore, only a small portion of necessary objects is transferred to prolog interpreter for further implementation. This specific searcher can readily solve the bottleneck of prolog machine. This expert system itself can have enough intelligence to answer pathway/genome problems, and can be adopted for biological process, drug discovery and medical diagnosis research. This type of prolog machine can also be applied to process huge knowledge bases rapidly for robotics and fifth generation computers.

1. Introduction

As genomic information becomes available for a growing number of organisms, it becomes essential to have efficient method to process knowledge resources for every organism. The biocyc collection of pathway/genome databases [1] contains more than 160 databases, including: ecocyc and metacyc. The pathway tools software [2] can be used to query, visualize, and analyze existing pathway/genome databases from the biocyc collection, and to create new databases for an annotated genome.

The main purpose of this paper is to develop an intelligent system for biological process, drug discovery and medical diagnosis research through pathway/genome databases. A specific prolog machine which is incorporated with a parallel hardware searcher is proposed in this paper. The pathway/genome databases and the programs to process these databases are represented as rules and facts. These horn clauses are treated as objects and stored in DRAM to increase access time. Each record of index files for these objects contains search attribute and physical address. Search module is a parallel hardware searcher with 9 processor units and 9 corresponding DRAM modules. Index files of object databases can be efficiently processed in search module. Java is used to integrate search module and SWI prolog interpreter [3, 4]. Search module finds the goal related objects through index files. Since only a small portion of objects is transferred to prolog interpreter for further procession, this searcher can readily solve the bottleneck problem of prolog machine. This expert system itself can have enough intelligence to answer gene network problems.

In 1980's, Japanese proposed the concept of intelligent computers called fifth generation computers, since they posses the superior robotic technology. Prolog is expected to be utilized as major language in this project. This hardware searcher proposed in the paper can provide fast search among huge data volume. Such search capability is desperately needed in the deduce process of prolog machine. The most significance of this searcher is trying to provide the key technique which can overcome the bottleneck of next generation intelligence computer systems. Therefore, computers can be upgraded to "electronic brains".

Since earlier computer developing stage, many researchers had been devoted to study different search algorithms. These algorithms are quite mutual. The cost of B tree index structures [5, 6, 7] is depending on the height of tree. B+ tree [8, 9] is a dynamic multi-level index file which is designed for very large files. But B+ tree is a one-dimensional access method. Search performance can be improved by using parallel technique and tree maintaining cost is too high.

The distributed and parallel architecture becomes the trend of computer technology when double core processor is popular in the market. Parallel search algorithms, such as R tree [10, 11, 12] and other

§ The design concept of this searcher had been utilized to apply the patent of Taiwan (application No: 095108044) and the patent of USA (application No: 11/416, 229) with financial support of Chung Hua University.

algorithms [13, 14, 15, 16], had been explored. But they still have overhead on maintaining tree and performance improvement is limited. Because, it is difficult to allow all processors involving on searching distributed data in each search step. The complexity and overall cost for this kind of hardware searcher is just too high to build.

In the proposed parallel hardware search machine, one of m processors has to enter into a rest state. The performance of this system is almost unaffected when m is becoming large. However, each processor can only process data in his memory location, and virtual search tree can be adopted in this machine. The cost of maintaining tree can be eliminated completely. The hardware of searcher can be easily designed by using FPGA chips. The performance of searcher can be dramatically increased at low production cost.

The remaining paper is divided into three main sections. Research projects of biocyc are introduced in Section 2. The pathway/genome expert system is designed in Section 3. The design of this parallel hardware searcher is given in Section 4.

2. Research projects of biocyc

The biocyc pathway/genome databases contain more than 160 databases, including: ecocyc, metacyc. The ecocyc is a model organism database for *Escherischia coli* K-12. The ecocyc project has integrated information on the *E. coli* genome, and on the *E. coli* metabolic and genetic networks from more than 11,000 publications. The metacyc is an encyclopedia of experimentally elucidated metabolic pathways and enzymes derived from 450 organisms.

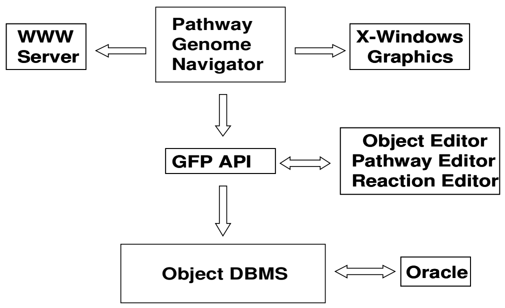


Figure 1. The overall architecture of biocyc.

Ocelot object database management system is utilized to manage pathway/genome objects which are named as frames. Common lisp is used as the native language of pathway tools. Bio-Loader is the interface program between lisp and databases. Many utility functions are developed by using lisp. Pathway tools provides querying capabilities, visualization tools such as for drawing pathways and genome maps, and interactive editing tools to allow users to update data such as modifying a metabolic pathway or defining a new DNA binding site for a transcription factor. The visualization and querying capabilities allow PGDBs to be published on the Web, or to be accessed as a desktop application on a PC.

3. Design a gene network expert system using a specific prolog machine

The overall architecture of the expert system is described in Section 3.1. Knowledge representation for gene network databases and programs is discussed in Section 3.2. In Section 3.3, several commands are implemented in the search module. The advantages of prolog for implementing PGDB are given in Section 3.4.

3.1 The overall architecture of the expert system

The overall architecture of our specially designed expert system is exhibited in Figure 2. Java language is used as API to integrate SWI prolog interpreter and search module. The gene network databases are represented as prolog facts. All the programs that process gene network databases are represented as prolog rules and facts. These horn clauses are treated as objects. Since recent cost reduction of DRAM, it is reasonable to store all objects into DRAM in order to increase data access time. The main functions in each block are explained below:

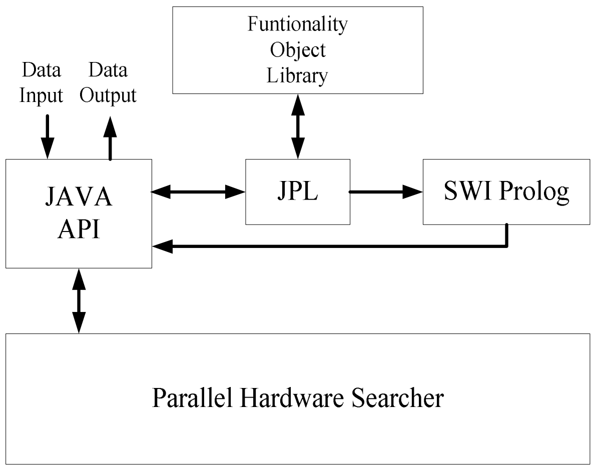


Figure 2. The overall architecture of our prolog machine.

1. **Java API:** Java language can be embedded with assembler language. Hence PC can directly store objects into or access objects from DRAM. Java can also input data files, goals or functionalities to hardware module, and display text output file or graphic data from prolog to GUI. PC can send a number of commands to searcher using interface addresses. These commands are: open pc and searcher connection, close pc and searcher connection, search, insertion, deletion, output result.

2. **SWI prolog:** SWI-Prolog has become a popular free software implementation of the Prolog language. It is formerly known as SWI at the University of Amsterdam. The programmer's environment has contributed most to its popularity. Many expert software developers adopt SWI prolog system and expend effort on large portable prolog applications where scalability, interfaces and networking are often important characteristics.

3. **JPI and functionality object library:** JPL is the interface program between search module and SWI prolog interpreter. All the prolog objects of horn clauses which are related to input goal are rapidly searched and collected from DRAM using hardware searcher. This set of horn clauses can be sent to SWI prolog interpreter to perform, can also be store in functionality object library. Therefore this functionality can be directly accessed from library and performed in SWI interpreter.

4. **Search module:** The architecture of parallel hardware searcher is exhibited in Figure3. There are 9 Processor Units (PUs) and 9 corresponding modules in the interface card of the search module. A number of connection lines are constructed for the communication among PUs. Each PU is corresponding with one giga byte DRAM. Only a small amount of qualified data needs to transfer to SWI prolog for further procession. The performance of the system can be dramatically increased to solve the time consuming problems in implementing huge and complex gene network databases.

3.2 Knowledge representation for PGDB and relevant programs

When PC opens connection with searcher, PC can store or retrieve gene network databases which are distributed in 9 DRAM modules. Gene network databases, including gene regulation network, signal transduction network, metabolic network and protein interacting network databases, can be stored in DRAM as prolog fact objects.

This object database system can contains databases from a number of organisms. The object databases of each organism contain several tables, and each table can have several index files which are convenient for data retrieval. The database dictionary contains the information of organism name, search attributes in each table, the size of each table (N), maximum loop number ($\lceil \log_{m-1} N \rceil$) and starting address of index file. It is noted that index file is a binary relation that contains two attributes: search attribute and object identifiers (can be represented as physical address). Multiple value attribute and referential information can be represented as an array of object identifiers.

Many software programs can be written in prolog rules and facts, such as: documentation cluster, predicting pathway, predicting hole filler and operon, applying gene network for different research purposes. This type of objects is combined into a control database file. The index file for these databases which contains prolog function name and physical address can be constructed for fast retrieval.

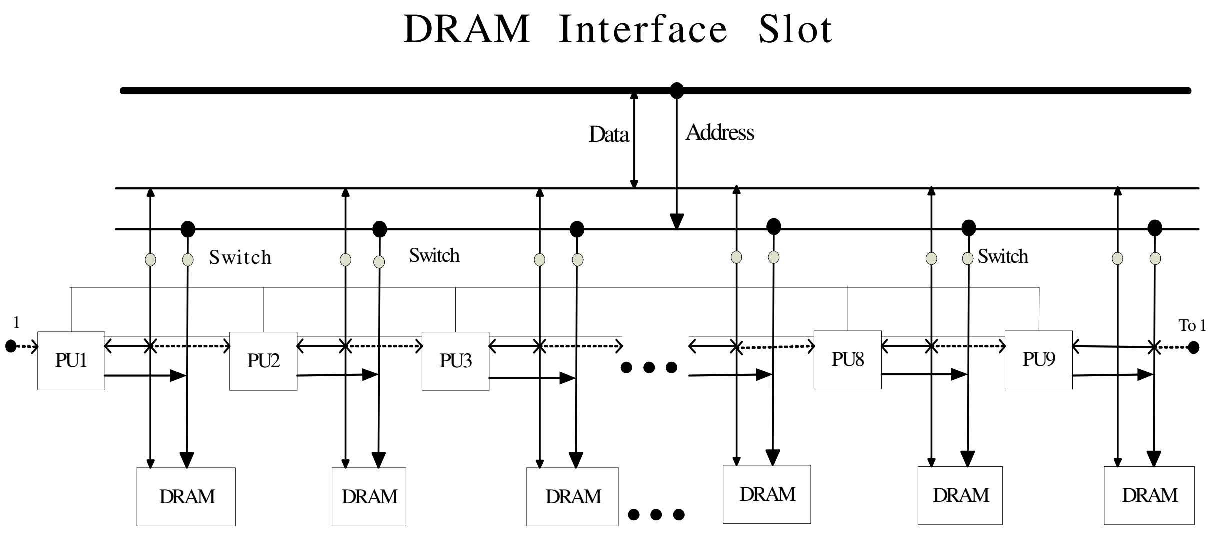


Figure 3. The design of a parallel hardware searcher.

3.3 Implement commands in search module

The search command implementation steps are described below:

1. PC opens the connection with searcher.
2. PC sends search command and three consecutive 64-bit data to searcher. Two 64-bit data for search criterion, and another 64-bit represents starting address of index file, maximum loop number.
3. PC closes the connection with searcher, and start to enter search loop.
4. The hardware of each PU performs the algorithm discussed in next section. Search operation can be completed within maximum loop number.
5. Searcher interrupts PC after completing the search operation.
6. PC sends result command, then PU that finds the required search value will transferred object physical address, object size and qualified object number to data bus. If no object is found, PU outputs the address for insertion.

To implement insertion, search the insertion value first. If criterion is found, physical address to insert will be calculated and reported by using count value of same function name. Otherwise, a PU will reports insert address.

3.4 The advantages of prolog language

Several advantages of prolog program are:

1. Prolog has strong capability to represent knowledge contained in natural language. Document clustering can be well performed by prolog.
2. All the other language use interface program to access databases. Databases are represented as prolog facts. Hence databases are a part of program and data access and processing is much easier.
3. Instead of writing all the detailed program steps, only logic rules are required in prolog. It will reduce the program developing cost, and has good capability for implementing complex programs.
4. Program size is greatly reduced, comparing with program size of other language.
5. The prolog program is considered as knowledge bases of expert system. The system using prolog has better intelligence than the system developed using other language.

The major disadvantage of prolog program is that this program requires fast searching capability over huge amount of knowledge bases. Parallel hardware searcher can be specially designed to overcome such disadvantage.

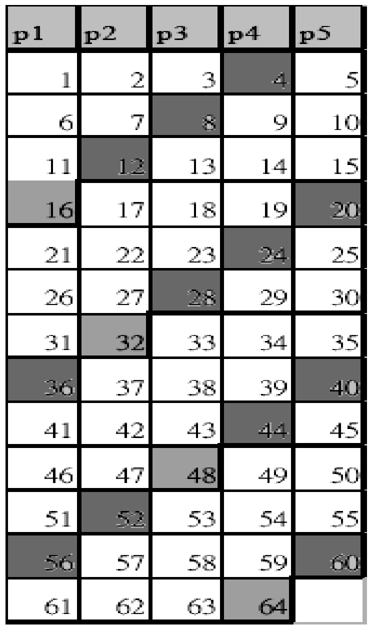
4. Design a parallel hardware searcher

Search is implemented by using an example in Section 4.1. Insertion and deletion is processed in Section 4.2. Search algorithm is given in Section 4.3.

4.1 Implement search operation

Let N represent the data size and m represent total processor number. N data is ordered according to search key values and distributed to m processors' memories. N is 64 and m is 5 in the example shown in the Figure 4. It is noted that each Location k (where k is between 1 and 64) contains a search key value and k is merely sequence ordered number. Pi is the i-th processor where i is between 1 and m. Therefore, the relationship among k, m and i is found: k MOD m = i. In another words, if we want to know the search key value of location k, then this value can be found in the memory of processor Pi.

The search key values of data can be sorted and distributed by using a hardware sorter. At initial, host sends BlockSize = 64, UpperBound = 64, m = 5, height = $\lceil \log_{m-1} N \rceil$ = 3 and path = 4 to all processors. This search example can be completed in 3 levels with 3 comparisons in worst case. If search value is found earlier by a processor, this processor will broadcast stop signal to all processor. Assume the search criterion can be found at Location = 38. The proposed algorithm can be described on detail below:



p1	p2	p3	p4	p5
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	

Figure 4. Distributing ordered data into multiple processor units for searching.

1. At level 1, all processor will process BlockSize = BlockSize / (m-1) = 16, UpperBound = 64 and UpperBound MOD m = 4. Therefore, all processor know P_4 represents path = 4 with location = 64. Since level number is odd, processor numbers must be increasing when corresponding path numbers are increasing. Each processor can calculate his representing path and the location to retrieve data. In Figure 5, P_1~P_4 represents path 1~4, and P_5 must take a rest (represents path 0). P_1~P_4 will retrieve data at

location 16, 32, 48 and 64. Finally P_3 finds that the search criterion is in his range, and broadcast path = 3 to all processors.

2. At level 2, BlockSize = 4, UpperBound = 48 and UpperBound MOD m = 3. P_3 represents path = 4 with location = 48. Since level number is even, processor numbers must be decreasing when corresponding path numbers are increasing. Each processor can calculate his representing path and the location to retrieve data. In Figure 5, P_1, P_5, P_4, P_3 represents path 1~4, and P_2 must take a rest. P_1, P_5, P_4 and P_3 will retrieve data at location 36, 40, 44 and 64. P_5 finds that the search criterion is in his range, and broadcast path = 2 to all processors.

3. At level 3, BlockSize = 1, UpperBound =40 and UpperBound MOD m = 0. P_5 represents path = 4 with location = 40. Level number is odd again. In Figure 5, P_2, P_3, P_4 and P_5 represents path 1~4, and P_1 must take a rest. P_2, P_3, P_4 and P_5 will retrieve data at location 37, 38, 39, and 40. P_3 finds that the search criterion at location 38, and broadcast a stop signal to all processors.

4.2 Implement insertion and deletion

Assume communication links are existent between every two adjacent processors. Hence, data can rotate left or right among m processors. To implement INSERT operation, search and find the location to insert is Location = d. All processors must rotate data right one location from Location = N to Location =d. Then insert the data into Location=d. The detailed algorithm for all processor P_i (where1 <= i <= m) is given in the following section. To implement DELETE operation, search the data for deleting and find the location to delete is Location = d. All processors must rotate data left one location from Location d+1 to Location N.

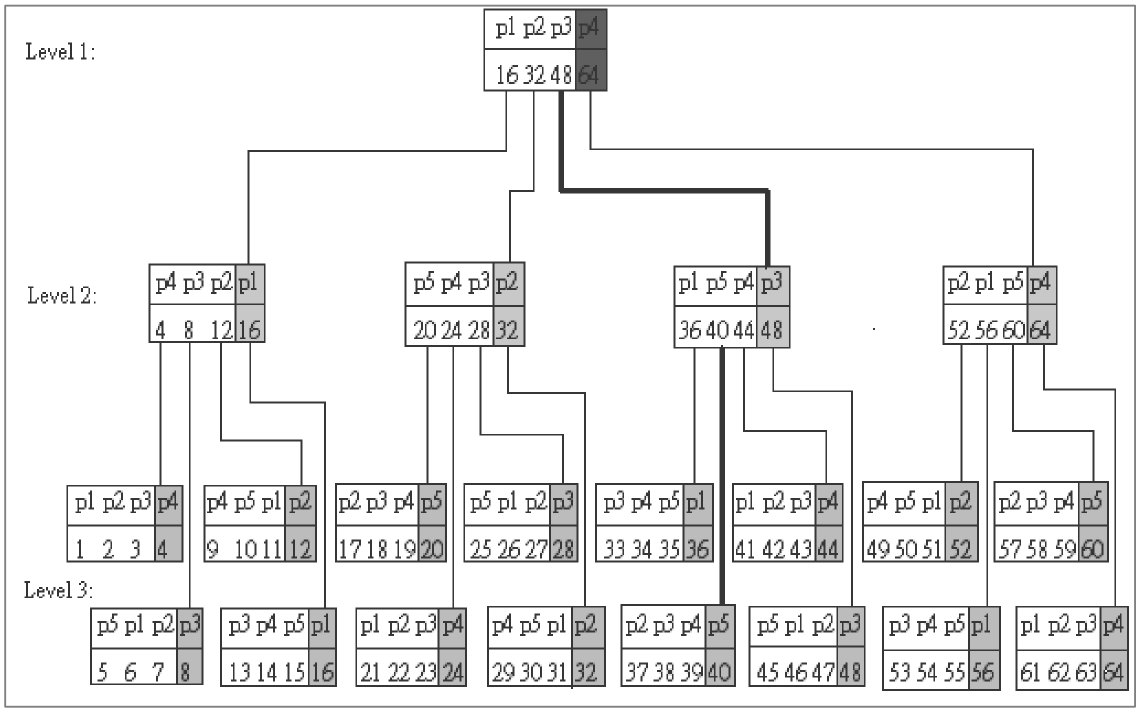


Figure 5. The virtual tree of parallel hardware searcher.

4.3 The virtual tree parallel search algorithm

```
/* At beginning, host sends N = 64, m = 5, HEIGHT = ⌈log_{m-1} N⌉ = 3 to all processors. */
BlockSize = N;    /* Set up initial values. */
UpperBound = N;
DEC _ m = m-1;
PATH = Dec _ m ;
LEVEL = 1;
/* Start to perform the operations in each level. */
while ( LEVEL <= HEIGHT )
{
  UpperBound = UpperBound - BlockSize *
               ( DEC_m – PATH );
  j = UpperBound MOD m;
/* Find the processor P_j represents PATH = DEC_m. */
  BlockSize = BlockSize / DEC _ m;
/* New block size is used in this level. */
  if ( Level is odd )
    {
      if ( i <= j ) {PATH= i +Dec _ m – j; }else {PATH
               = i – j –1; }
    };
```

```
        else {
                if ( i >= j ) {PATH = Dec _ m + j - i; }else
                        {PATH = j – i –1;};
                };
 /* Pi represents path number in PATH. */
        if ( Path ==0 ) {Pi takes a rest; }
        else { Location = UpperBound – BlockSize *
                              ( Dec _ m – path );
                };
/* P_i gets data from Location and compare with search
value. P_i sends STOP signal when criterion is found. If
P_i finds that criterion is located in his range, then P_i
broadcast his path to all processors. */
     Level = Level+1;
};
```

5. Conclusion

A gene network expert system had been designed using a prolog machine that is incorporated with a parallel hardware searcher. Gene network databases were represented as facts, and the programs were presented as both rules and facts. All the facts and rules are treated as objects and stored in DRAM to increase access time. Java has been used to integrate the whole system. Only a small amount of qualified data needs to transfer to PC for further procession. This searcher can readily solve time consuming problem in prolog system. This expert system itself cam have enough intelligence to answer gene network problems.

6. Reference

[1] R. Caspi, and etc., “MetaCyc: A multiorganism database of metabolic pathways and enzymes”, *Nucleic Acids Res*, Volume 34. pp. 511-516, 2006.

[2] P. Karp, S. Paley, and P. Romero "The pathway tools software", *Bioinformatics,* Volume 18. S225-S232 2002.

[3] T. Schrijvers, J. Wielemaker and B. Demoen, “Constraint handling rules for SWI-prolog”, *Workshop on (Constraint) Logic Programming*, Ulm, February, 2005.

[4] J. Wielemaker and A. Anjewierden. “An architecture for making object-oriented systems available from Prolog”. *In Alexandre Tessier, editor, Computer Scienc, abstract 2002.* http:// lanl.arxiv.org /abs /cs.SE /0207053.

[5] R. BAYER and C. MCCREIGHT, "Organization and maintenance of large ordered indexes", *Acta Inf.* Volume 1, No.9, pp. 173-189, 1972.

[6] D. Lomet. " The evolution of effective B-tree: page organization and techniques: A personal account", *ACM SIGMOD Record*, Volume *30(3), pp. 64-69, Sep.* 2001.

[7] T. Johnson and D. Sasha, "The performance of current B-tree algorithms", *ACM Transactions on Database Systems (TODS)*, Volume 18 , Issue 1, pp. 51-101, March 1993.

[8] S. Chen, P.B. Gibbons, T.C. Mowry and G. Valentin, "Fractal prefetching B+-Trees: optimizing both cache and disk performance", *Proceedings of the ACM SIGMOD international conference on Management of data*, 2002 , Madison, Wisconsin.

[9] S.W. Kim , H.S. Won, "Batch-construction of B+-trees", *Proceedings of the 2001 ACM symposium on Applied computing*, pp. 231-235, March 2001.

[10] A. Guttman "R-trees: A dynamic index structure for Spatial Searching", *Proceedings of the ACM SIGMOD*, pp. 47-57, June 1984.

[11] T. Sellis, N. Roussopoulos, and C. Faloutsos. "The R+ Tree: A dynamic index for multi-dimensional Objects", *Proceedings of the 13th VLDB Conference*, 1987.

[12] B. Wang, H. Horinokuchi, K. Kaneko and A. Makinouchi , "Parallel R-tree search algorithm on DSVM", *Proceedings sixth International Conference on Database Systems for Advanced Applications*, pp. 237-244, April 1999.

[13] S. Gerard and B. Chris, "Parallel text search methods", *Communications of the ACM*, Volume 31, Issue 2, pp. 202-215, February 1988.

[14] M. Sergey, R. Sriram, Y. Beverly and G.M. Hector, "Building a distributed full-text index for the web", *ACM Transactions on Information Systems (TOIS)*, Volume 19 , Issue 3, pp.217-241, July 2001.

[15] P. Sakti and H.K. Myoung, "Parallel processing of large node B-trees", *IEEE Transactions on Computers*, Volume 39, No.9, pp.1208-1212, September 1990.

[16] M.A. Torres, S. Kuroyanagi and A. Iwata, "A fast parallel search method for large dictionaries", *Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems*, pp. 207 -214, 1998.