Parallel Dictionary Operations : Algorithms and Architectures

by

ATUL VARSHNEYA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A Thesis Submitted
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

to the

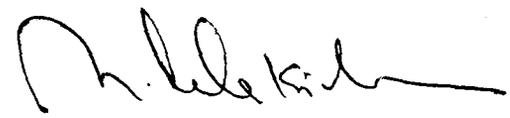
INDIAN INSTITUTE OF TECHNOLOGY, DELHI
NEW DELHI 110 016

SEPTEMBER 1995

Dedicated to Renu and Aniruddh

Certificate

This is to certify that the thesis entitled **Parallel Dictionary Operations : Algorithms and Architectures**, being submitted by Atul Varshneya to the Indian Institute of Technology, Delhi, for the award of Degree of Doctor of Philosophy, is a bonafide research work carried out by him under our supervision and guidance. The results obtained in this thesis have not been submitted to any other University or Institute for the award of any other degree or diploma.



B.B. Madan
Professor, (on leave)
Department of Computer Science
and Engineering,
Indian Institute of Technology,
New Delhi 110016

M. Balakrishnan
Associate Professor
Department of Computer Science
and Engineering,
Indian Institute of Technology,
New Delhi 110016

Acknowledgement

I am extremely grateful to my supervisors, Prof. B.B.Madan and Dr. M. Balakrishnan for providing invaluable scientific guidance, encouragement and moral support ever since I know them, and especially throughout the duration of the thesis work. They are extremely helpful and warm persons , and both of them went out of their ways to guide me at odd times too. Their probing comments and insightful suggestions have made this work possible. I have personally been benefitted both academically and otherwise by working in their guidance.

My thanks are also due to Prof. S.N.Maheshwari, Drs. Subashish Banerjee, Shashi Kumar, Sandeep Sen, Sanjiv Kapoor, and Sandeep Sen for several useful technical discussions and their comments.

My parent have always encouraged and supported all my endeavors in life. Without their blessings my attempts would not have borne fruits. To them I give my greatest thanks.

My work would not have been possible without the love, encouragement and support of my wife Renu and my son Aniruddh. The innumerable sacrifices on their part can hardly be narrated.

Atul Varshneya

(Atul Varshneya)

Abstract

Dictionary data structures are very important structures in design of algorithms, and find much use in various applications such as sorting, searching symbol-table, and decision table implementations. Such a data structure consists of a set S of data elements, each of which is composed of a key-record pair $K = (k, r)$, where k is the key used for searching the data element K, and r is the associated record or a pointer to the record stored in a different set. A dictionary machine is required to support the following instructions: Insert, Delete, and Member. Insert adds a new element to S, Delete removes an existing element from S, and Member determines if a certain element belongs to S.

Various solutions have been proposed to parallelise the operations of dictionary machines. these comprise of solutions in form of machine architectures designed to perform dictionary operations efficiently, and of algorithms that allow concurrent access to search data structures. It has been shown that if a dictionary operation is executed by a number of processors in parallel, then the best speed-up possible is logarithmic in the number of processors. This renders such an approach unattractive. The approaches, therefore, have been to increase the total throughput in dictionary operations by increasing the number of instructions that can be handled by the machine at the same time rather than to speed-up individual instructions.

The work reported in this thesis first presents two methods based on general purpose computing platforms for multiple accesses to dictionary search structures. Next, certain characteristics of the algorithms for dictionary operations are noted and a new processor coupling architecture, MC machine, is presented on which those can be implemented efficiently. The presented architecture is shown to be useful for a number of other engineering and

scientific applications. Finally, a dictionary machine based on MC machine is presented to show the simplicity of implementation and efficiency of its performance on this architecture.

Most of the work in parallel dictionary operations has been done for single dimensional search problems. In this work we first present a lock-coupled solution for accessing a multi-dimensional search tree, viz., kHB-tree, concurrently by a number of processes. One important aspect of the approach followed is that it reduces the accesses in the tree only to a small closed portion of the tree, also it has another advantage of having a fixed upper bound on the total number of locks held by a process. The proposed solution is clearly applicable for AVL trees which are kHB trees with $k = 1$. The solution is shown to be simpler and providing some improvements over the solutions proposed for AVL trees.

Next, a method is presented to handle multiple accesses to B^+-tree over an array of message passing processors. The method utilises pipelined access to the tree by distributing the tree levels among the processors in the array. The main ideas here were to develop a dictionary machine based on general purpose computing platform, and to do away with the problem of memory access contentions faced in shared memory systems. The objectives achieved in this method are, firstly, achieving a coarse grained dictionary machine with a small number of processors connected in a simple topology (linear array). And secondly, performing load balancing among the processors along with the execution of dictionary operations which increases the pipeline interval only by a constant. This results in a an improvement in throughput which is linear in the number of processors. Timing measurements carried out by implementing this method verified the performance of this scheme.

Based on an observation about a typical characteristic of most search

structures, that the accesses are highly localised within the structure topology, we developed a processor coupling architecture — MC-Machines — which is tuned to such requirements. The processors are coupled by memory modules with multiple access ports. Thus adjacent processors in the machine can directly share data using this shared memory. This makes the operations in search structures very easy to distribute among processors as adjacent processors can be made to store adjacent tree nodes.

The architecture provides two key benefits. Firstly, it is scalable like message passing architectures, and secondly, it provides partial sharing of data among processors which makes programming paradigm on this similar to the one of shared memory machines. Although the architecture was developed primarily for dictionary operations, but is shown to be equally beneficial for a number of applications in areas of engineering and scientific computing.

Finally, a method for pipelined access to B^+-trees on MC-machines is also presented. This method provides a more efficient and simple scheme than the pipelined access method discussed above over an array of message passing processors. These improvements are facilitated by the presence of shared memory among adjacent processors. The basic idea used in this method is similar to the previous method, i.e., that of maintaining load balance among processors by distributing equal number of levels among processors. But a number of issues related to the tree re-structuring operations and data re-distribution for load-balancing get resolved in a much simpler and more efficient manner on this architecture.

Contents

1 Introduction 1

1.1 Background 2

1.2 Parallel execution of dictionary operations 3

1.2.1 VLSI dictionary machines 4

1.2.2 Dictionary machines on multiprocessors 11

1.3 Dictionary operations on multi-dimensional keys 17

1.4 Computer architectures for parallel dictionary operations . . . 20

1.4.1 Problem-specific versus general purpose 20

1.4.2 General purpose parallel computing architectures . . . 21

1.5 Overview of this work 22

1.6 Organization of the thesis 26

2 Concurrent operations on kHB-trees 28

2.1 Introduction 29

2.2 Rebalancing operations on kHB-trees 32

2.3 Concurrent operations on kHB-trees 33

2.3.1 The locking mechanism 36

2.3.2 Search process 38

2.3.3 Insert process 39

2.4 Proof of correctness 42

2.5 Performance 47
2.6 Delete operation in kHB-trees 49
2.7 Conclusions 53

3 B^+-tree over an array of processors **55**
3.1 Introduction 56
3.2 Strategy 58
3.3 B^+-tree operations 62
3.3.1 Insertion in a B^+-tree 63
3.3.2 Deletion in a B^+-tree 64
3.3.3 Top-down algorithm 64
3.4 Data structures and messages 66
3.4.1 Data structures 66
3.4.2 Messages 68
3.5 Data re-distribution mechanism 72
3.5.1 Achieving the quiescent state 73
3.5.2 Determining the direction of re-distribution 75
3.5.3 Data structure manipulations 78
3.6 The complete picture 79
3.6.1 Insert 82
3.6.2 Delete 84
3.7 Performance analysis 85
3.8 Conclusions 91

4 MC-Machines **95**
4.1 Introduction 96
4.2 Memory Coupled architecture 98
4.2.1 Memory sharing topologies 99

4.2.2 Message passing between processors in Memory Coupled systems . . . 101
4.3 Implementation issues . . . 101
4.3.1 Implementing the shared memory . . . 103
4.3.2 Synchronization and atomic access primitives . . . 105
4.3.3 Address mapping of shared memory . . . 107
4.4 Programming the MC machine . . . 109
4.5 Applications on MC-machine . . . 112
4.5.1 Image processing applications . . . 113
4.5.2 Particle dynamics simulations . . . 114
4.5.3 Partial differential equations . . . 114
4.6 Simulator . . . 116
4.7 Conclusions . . . 118

5 B^+-tree on MC-machine 119
5.1 Introduction . . . 120
5.2 Overview of the approach . . . 120
5.2.1 The key ideas . . . 120
5.2.2 Advantages of implementation on MC-machine . . . 129
5.3 Implementation of B^+-tree on MC-machine . . . 130
5.3.1 Data Structures . . . 131
5.3.2 Data distribution mechanism . . . 133
5.3.3 Tree traversal mechanism . . . 136
5.3.4 Interaction between traversal and movement operations 138
5.3.5 The algorithm control structure . . . 141
5.4 Performance . . . 143
5.5 Conclusions . . . 147

6 Conclusions **149**

6.1 Introduction 149

6.2 Summary of the results 150

6.3 Suggestions for further work 152

A Algorithms for concurrent access to kHB-tree **154**

A.1 Search 155

A.2 Insert 156

A.2.1 Function to REBAL lock parent 156

A.2.2 Insert process 156

B Algorithms for pipelined access to B^+-tree using message passing **161**

B.1 Top-level program 162

B.2 Insert procedure 164

B.2.1 Main insert procedure 164

B.2.2 Insert traversal-step function 165

B.3 Delete procedure 167

B.3.1 Main delete procedure 167

B.3.2 Delete traversal-step function 169