Analyzing Cache-Oblivious String B-Trees Data Structure for Efficient IP Prefix Queries

Dr. Erin Wolf Chambers, Sagar Calnoor Rajashekar

What is *Cache-Oblivious* algorithm?

An algorithm is ***cache oblivious*** if no program variables dependent on hardware configuration parameters, such as cache size and cache-line length need to be tuned to minimize the number of cache misses. From [1], we known cache oblivious algorithms uses asymptotically optimal amounts of work, in order to move data asymptotically optimally among multiple levels of cache.

Why the need for *Cache-Oblivious algorithm* in *IP prefix lookup*? (problem statement)

Network Routers uses a technique called Forwarding. Forwarding is moving incoming packets to appropriate interface. Routers use forwarding table to decide which incoming packet should be forwarded to which next hop. Where IP prefix is a prefix of IP address plays important role in forwarding. Considering when all computers on one network have same IP prefix. For example, in 192.24.0.0/18, 18 is length of prefix and prefix is first 18 bits of the address that can be held as string of characters. So, forwarding in network routers basically look at destination address's IP prefix, searches the forwarding table for a match and forwards the packet to corresponding next hop in forwarding table. So, our problem here presents a need for efficient data structure for finding the longest prefix of a query string p in a dynamic database of strings. When the strings are IP addresses then this is the IP-lookup problem. Creating a need for data structure that is more inclined towards solving I/O efficiency.

In addition to good I/O performance an efficient data structure for IP-lookup should be able to perform queries at the line rate (which is about 100 Gbps and more today). The data structure has to be scalable since the number of prefixes a router has to maintain is growing rapidly as well as the length of these prefixes. Finally, we need to support fast updates mainly due to instabilities in backbone routing protocols and security issues. The rapid growth of the Internet has brought the need for routers to maintain large sets of prefixes, and to perform longest prefix match queries at high speeds [5]. A main issue in the design of routers is the size of the expensive high-speed memory used by the router for packet forwarding. One can reduce the size of this expensive memory by using external memory components. Hence, the need for cache oblivious data structures to enhance fast IP prefix lookup.

Problem statement considers the longest prefix problem which is defined as follows. The input consists of a set of strings S = {p1 . . . pn} which we shall refer to as prefixes. We want to preprocess S into a data structure such that given a query string q we can efficiently find the longest prefix in S which is a prefix of q or report that no prefix in S is a prefix of q. We target the dynamic version of the problem where we want to be able to insert and delete prefixes to and from S, respectively.

How do we plan to *address the above problem statement*?

Brodal and Fagerberg [3] derives a cache oblivious data structure for manipulating strings. This data structure could be used to obtain an I/O efficient solution for longest prefix problem. We plan to combine above knowledge with the string B-tree to obtain a general I/O efficient solution. Where we expect the solution to support a query with a string p using O (log B

$(n)+|p|/B)$ I/O operations, where B is the size of a disk block. It also should supports an insertion and a deletion of a string p with the same number of I/O's. The size of the data structure is linear in the size of the database and the running time of each operation is O(log(n) + |p|).

References:

[1]. Frigo, M., Leiserson, C. E., Prokop, H., and Ramachandran, S. Cache-Oblivious Algorithms in ACM Trans. Algor. 8, 1, Article 4 (January 2012)

[2]. M.A. Bender, E.D. Demaine, M. Farach-Colton, Cache-oblivious b-trees, in: IEEE (Ed.), 41st Annual Symposium on Foundations of Computer Science: Proceedings: 12–14 November, 2000, Redondo Beach, California, IEEE Computer Society Press, Silver Spring, MD, pp. 399–409.

[3]. G S. Brodal and R. Fagerberg. Cache-oblivious string dictionaries. In Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 581–590, 2006.

[4]. Cache-Oblivious Search Trees Project
http://supertech.csail.mit.edu/cacheObliviousBTree.html

[5]. W. Eatherton, Z. Dittia, and G. Varghese. Tree Bitmap : Hardware/Software IP Lookups with Incremental Updates. ACM SIGCOMM Computer Communications Review, 34(2):97–122, 2004