Two-level Mapping based Cache Index Selection for Packet Forwarding Engines

Kaushik Rajan† and R. Govindarajan†‡
‡Department of Computer Science and Automation
†Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
kaushik@hpc.serc.iisc.ernet.in
govind@csa.iisc.ernet.in

Abstract

Packet forwarding is a memory-intensive application requiring multiple accesses through a trie structure. The efficiency of a cache for this application critically depends on the placement function to reduce conflict misses. Traditional placement functions use a one-level mapping that naively partitions trie-nodes into cache sets. However, as a significant percentage of trie nodes are not useful, these schemes suffer from a non-uniform distribution of useful nodes to sets. This in turn results in increased conflict misses. Newer organizations such as variable associativity caches achieve flexibility in placement at the expense of increased hit-latency. This makes them unsuitable for L1 caches.

We propose a novel two-level mapping framework that retains the hit-latency of one-level mapping yet incurs fewer conflict misses. This is achieved by introducing a second-level mapping which reorganizes the nodes in the naive initial partitions into refined partitions with near-uniform distribution of nodes. Further as this remapping is accomplished by simply adapting the index bits to a given routing table the hit-latency is not affected. We propose three new schemes which result in upto 16% reduction in the number of misses and 13% speedup in memory access time. In comparison, an XOR-based placement scheme known to perform extremely well for general purpose architectures, can obtain upto 2% speedup in memory access time.

Categories and Subject Descriptors: C.2.1 :Network Architecture and Design
General Terms: Design, Performance, Experimentation.
Keywords: Network Processors, Cache Architectures.

1. INTRODUCTION

The ever increasing demand for network bandwidth and the need to support sophisticated streaming applications requires routers to process packets at higher line rates. Due to the memory intensive nature of packet forwarding, modern day routers are constrained by the performance of the memory system rather than the processing capability. One of the key functionalities of routers is to forward incoming packets through an appropriate output port. This involves looking in a table of prefixes, finding the longest prefix match (LPM) for the destination IP-address, and using the corresponding output port.

Most algorithms used for LPM make use of a trie data structure to store the contents of the routing table [18][3]. Even with efficient lookup algorithms, a single IP-lookup incurs a significant number of memory accesses. For example, in the worst case, the LC trie algorithm [14] requires 7 memory accesses and the Lulea algorithm [2] requires as many as 12 memory accesses. With the requirement to process packets at wire speeds (10Gbps or higher), a router has to process more than 20 million packets every second. Under such requirements supporting IP-lookup stresses the memory system greatly.

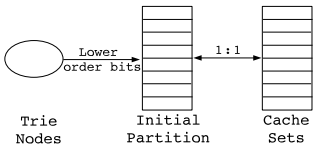
Routers typically have a multi-threaded multiprocessor architecture. Caching is a technique orthogonal to multi-threading for reducing memory access overhead in network applications. While multi-threading hides memory latency and improves processor utilization, caching reduces the number of accesses that go to memory and reduces average memory access time. It improves performance by reducing the contention for memory, preventing saturation of memory bandwidth and reducing the number of context switches. Earlier work suggests that multi-threading alone is not sufficient to ease the memory bottleneck. Caching can further improve the throughput by upto 200% [12].

Various proposals have been made to use caches to speedup the lookup algorithm [1][4][5][16][20]. However, it has been observed that in packet forwarding applications the miss rate can be as high as 75% because of poor locality patterns and cache conflicts. Thus, any reduction in misses should significantly improve the average case performance of the router. Earlier studies reveal that there is more locality in trie node accesses than in IP address accesses. Based on this Low et. al. propose a single cache for the nodes of the trie [1]. The Heterogeneously Segmented Cache Architecture (HSCA) [16] further improves the performance of trie node caches by separating out the cache for nodes that are direct descendants of the root node (Level-One/LO nodes) from that for Lower-Level (LL) nodes.

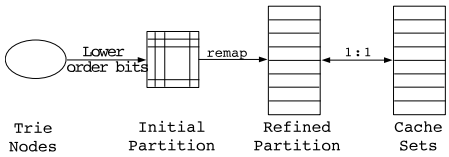
It is generally observed that the trie structure used for forwarding typically employs a large branching factor (2^{16}) for the root node [18] thus introducing a large number of *superfluous LO nodes* (nodes which do not contain any information but get added when a using fixed root branching factor, example in Section 3). For the routing tables we use,

PACT'06, September 16–20, 2006, Seattle, Washington, USA.

a branching factor of 2^{16} results in as many as 75% superfluous LO nodes. As these superfluous nodes are interspersed with useful nodes, using a conventional placement function (indexing with lower order bits) for the LO cache may-not uniformly distribute the useful nodes among the sets. The



(a) Base Mapping



(b) Two-level Mapping

Figure 1: Mapping from nodes to sets

mapping achieved by conventional placement functions is depicted in Figure 1(a). In these mappings, the set of useful LO nodes are partitioned based on the lower order *log m* bits into m partitions. These m partitions are then mapped one-to-one to m cache sets. Both the partitioning and the mapping of partitions to sets is *rigid*. While this rigidity in the partitioning and mapping helps to achieve efficient direct indexing and hence low hit-latency, we observe that it results in an uneven distribution of useful LO nodes among partitions. Other placement schemes, like those based on XOR functions [7], are also restrictive as they use a predetermined function of the address bits to form the index. Alternative proposals such as software managed full associativity [8] and V-way cache [15] proposed in the context of general purpose processors, reduce conflict misses by flexibly mapping the accessed data to cache lines. However, this flexibility is attained through indirection which results in increased hit-latencies.

There are two main contributions of this paper.

- *A two-level mapping framework which achieves the desired flexibility to reduce conflict misses.* The framework (refer to Figure 1(b)) maintains a rigid partitioning of useful LO nodes into initial partitions (IP), but reorganizes the nodes in the IPs into more evenly populated refined partitions (RPs) through intelligent cache index selection. The RPs are then rigidly (one-to-one) mapped to cache sets. By introducing an additional layer of flexible mapping between IP and RP, a better distribution of LO nodes to sets can be achieved.
- *Three schemes to intelligently remap the nodes in IPs into RPs.* (i) In FLEX, the remapping is done by expanding each of the IPs into two RPs by using a higher order address bit that is most appropriate for that IP. (ii) In IMAP we reorganize the IPs by combining 2 IPs together to form a single RP. (iii) In V-ASSOC, each RP is assigned either a high associativity or a low associativity based on the number of nodes mapping to it. V-ASSOC is basically a two-level mapping based variable associativity cache. In each of these schemes the remapping used for one IP can be different from that of another. This provides us with a flexibility to adapt the node to set mapping to a given routing table. We propose simple hardware implementations to support FLEX, IMAP and V-ASSOC remapping schemes. The implementations do not increase cache access time making them attractive solutions.

We evaluate the performance of our schemes and compare them with the base LO cache [16], the bit-selection scheme (BS) [16] (Section 3.1.2 explains the algorithm) and an XOR based placement scheme [7]. The XOR based placement scheme forms a k-bit index by XORing two k-bit pairs from the address. Such a placement function has been shown to perform extremely well for general purpose processor caches. In our experiments the lower order k bits of the address (A_{k-1}, A_0) are combined with the k bits (A_{15}, A_{15-k}). This combination is chosen because the address of the 2^{16} LO nodes can only differ in the lower order 16 bits. Experimental results indicate that our schemes achieve a better node to cache set mapping. Further, FLEX and IMAP can reduce the number of misses by 16% and achieve memory access speedup of upto 13%. In comparison, the BS scheme barely improves performance, while XOR based placement achieves at best a 3% reduction in misses and 2% speedup in memory access time.

The rest of the paper is organized as follows. Section 2 provides the necessary background and reviews the related work. Section 3 motivates the need for two level mapping with the help of an example. In Section 4 we describe the hardware implementation. Section 5 presents the simulation methodology and introduces the traces used for performance evaluation. In Section 6 we report the experimental results comparing the various schemes. We conclude in Section 7 with a summary of the results and suggest future enhancements.

2. BACKGROUND AND RELATED WORK

2.1 Forwarding using Tries

In order to facilitate LPM routers typically represent the routing table information in the form of a trie [3][18]. The trie is traversed based on the bits of the incoming packet's destination IP-address. The various trie based schemes use a compact representation of the simple binary trie and differ only in the manner in which they compress the trie [3][18]. One common compression done in almost all trie based algorithms is to use a large root branching factor (usually 2^{16}). This is because most routing tables contain very few prefixes of length less than 16. An efficiently compressed trie structure commonly used in earlier cache studies [1][16] is the Level Compressed trie [14][17]. An LC-trie is laid out in an array (refer to Figure 2), with nodes closest to the root (LO nodes) preceding the remaining nodes (LL nodes) in the array. Once the trie is laid out in this way the index of the array determines the memory address in the memory space [16]. The size of a node can vary from 4 to 8Bytes and the next-hop information can be stored either along with the trie nodes [17] or in an additional auxiliary table[1] [14].

[1]These contain information regarding the nexthops.

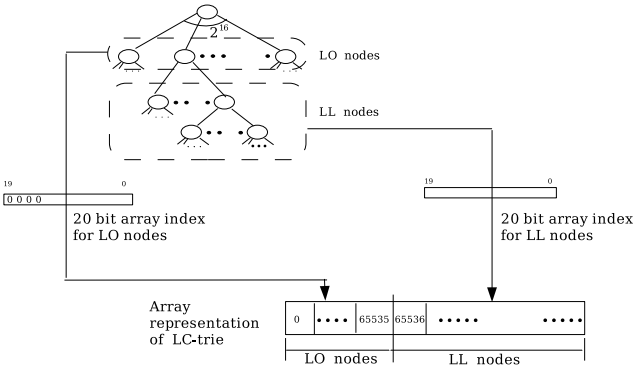


Figure 2: LC-trie and its array representation

2.2 Related work

Mudigonda et al [12] published an interesting study comparing various techniques for reducing the memory performance bottleneck. The study reveals that a combination of multi-threading and caching is essential for a balanced NP architecture.

Several proposals have been made for the usage of caches to speed up forwarding [1][4][5][16][20]. These include caching schemes that cache the IP addresses or prefixes and those that cache trie nodes. Notably, Chieueh and Pradhan [5] introduce Intelligent Host Address Range Caching (IHARC), which first converts prefixes into distinct address ranges and then combines non-adjacent ranges sharing a common output port. This work is extended by introducing additional splitting bits which facilitate a variable cache set mapping, where, partitions to which more number of address ranges map, are given more sets [4]. Recent research focuses on caching the data structures used for forwarding [1][16]. The additional locality in accesses to the nodes of a trie data structure used for forwarding is exploited by using a cache to store the recently accessed nodes of an LC-trie [1]. The authors of HSCA [16] observe that LO nodes are accessed more often than LL nodes. They also note that the locality among the LO nodes is much more than the locality among LL nodes. Both these observations are based on the fact that an access to any of the leaf nodes has to necessarily pass through an LO node, only a few of which are useful. Based on these findings they propose a segmented cache organization with a relatively large cache for LO nodes and a much smaller cache for LL nodes. By skewing the cache sizes in favor of the LO cache they are able to give more importance to the LO nodes.

All these caching schemes share a few common features. Firstly, caches proposed for forwarding accommodate exactly one entry per cache line. This is because of the total lack of spatial correlation between successively accessed prefixes. Secondly, every-time the entries of the routing table in-use change, the cache entries become stale and need to be updated/flushed. However, even-though route updates can happen frequently, the updates are made to a backup routing table which is copied to the in-use table much less frequently [11][3].

A major drawback of the above mentioned caches is that they use a one-level mapping based placement function. Our proposed two-level mapping framework differs from these schemes in that we make use of the knowledge of the distribution of useful nodes to improve cache placement.

Among enhancements proposed for general purpose processor caches, XOR based placement schemes [7] uses the information content of more address bits than the conventional lower order bits to identify the cache index. Kharbutli et. al. [10] introduce two new hash based indexing schemes that try to eliminate the worst case conflict behaviour of L2 caches. An Indirect Index Cache (IIC) [8] attempts to achieve full associativity with the aid of software management at the expense of a variable cache access time. Lastly, in the V-way cache organization [15], the associativity of various cache sets of an L2 cache is dynamically adjusted according to the demands of the application. But, this flexibility is introduced at the expense of an increased access time. Our proposed two-level mapping based index selection schemes are able to reduce the potential for conflicts without incurring additional hit time penalties.

3. MOTIVATION

In this section the drawbacks of one-level mapping based cache indexing are illustrated with the help of an example. Then the arguments in favor of two-level mapping are put forth and the motivation for the proposed IP to RP remapping schemes is provided.

Consider the routing table in Table 1. The table has 16 entries with a maximum prefix length of 7. Now if we consider a root branching factor of 2^7, the number of useful nodes contributed by each entry is shown in the Table. These are the only useful LO nodes in the trie. Hence out of a total of 128 LO nodes there are 32 (25%) useful nodes and the remaining 96 (75%) nodes are superfluous. If these useful nodes were to be divided into 8 sets we would ideally want each set to have exactly 4 nodes.

Sno	prefix	prelen	next-hop	LO nodes
1	0001111	7	3	1
2	010000	6	3	2
3	0100010	7	2	1
4	01001	5	1	4
5	0110001	7	4	1
6	0111111	7	2	1
7	100000	6	1	2
8	10001	5	1	4
9	100100	6	2	2
10	1010010	7	5	1
11	110110	6	3	2
12	1101111	7	4	1
13	11100	5	1	4
14	1110100	7	2	1
15	1110111	7	4	1
16	11110	5	2	4

Table 1: Routing Table

3.1 One-level mapping based placement

3.1.1 Conventional Mapping

The simplest way to map the nodes into cache sets would be to partition the useful nodes based on the lower order bits. Unfortunately, this naive partitioning will result in an uneven distribution of useful nodes among sets. For the example in Table 1, a 3 bit index would form eight sets with 5, 6, 4, 2, 4, 3, 2, and 6 nodes respectively as shown in Table 2. Clearly the mapping is not ideally suited for caching. The

IP_0	IP_1	IP_2	IP_3	IP_4	IP_5	IP_6	IP_7
0 **1** 0 0 0 0 0	0 **1** 0 0 0 0 **1**	**1 1 1** 0 0 **1** 0	**1 1 1** 0 0 **1 1**	0 **1** 0 0 **1** 0 0	0 **1** 0 0 **1** 0 **1**	0 **1** 0 0 **1 1** 0	0 **1** 0 0 **1 1 1**
1 0 0 0 0 0 0	0 **1** 0 0 0 0 **1**	0 **1 1** 0 0 **1** 0	**1 1 1 1** 0 **1 1**	**1** 0 0 0 **1** 0 0	**1** 0 0 0 **1** 0 **1**	**1** 0 0 0 **1 1** 0	**1** 0 0 0 **1 1 1**
1 1 1 0 0 0 0	**1** 0 0 0 0 0 **1**	**1** 0 **1** 0 0 **1** 0	-	**1 1 1** 0 **1** 0 0	**1 1** 0 **1 1** 0 **1**	-	**1 1 1** 0 **1 1 1**
1 0 0 **1** 0 0 0	**1 1 1** 0 0 0 **1**	**1 1 1 1** 0 **1** 0	-	**1 1** 0 **1 1** 0 0	-	-	0 0 0 **1 1 1 1**
1 1 1 1 0 0 0	**1** 0 0 **1** 0 0 **1**	-	-	-	-	-	0 **1 1 1 1 1 1**
-	**1 1 1 1** 0 0 **1**	-	-	-	-	-	**1 1** 0 **1 1 1 1**
5	6	4	2	4	3	2	6

Table 2: Initial Partitions formed using bits b_1, b_2 and b_3

highly populated sets would have more conflicts while the sparsely populated ones will be underutilized.

3.1.2 Node to Set Mapping with BS

The Bit Selection algorithm [16] is a simple extension of conventional indexing where, instead of using the lower order bits, an attempt is made to choose the index bits that partition the nodes most uniformly. The algorithm picks the bits one at a time, picking the next bit based on how well it, along with previously chosen bits, partitions the nodes. For the example under consideration, the 3 bits chosen would be b_0, b_1 and b_4. When these 3 bits are used, the resulting number of nodes per set is shown in Table 3. It can be observed that we still have two IPs (sets) with 6 nodes which are likely to end up having more conflicts than the four IPs (sets) with 3 nodes each.

		b_1b_0			
		00	01	10	11
b_4	0	6	6	3	4
	1	3	3	3	4

Table 3: Number of nodes per partition for 8 IPs formed using bits b_0, b_1 and b_4 chosen by BS

The reason for the non-uniform distribution in the above two schemes is that they are very restrictive in cache index selection, i.e., both use a fixed set of index bits to directly determine the mapping from nodes to cache sets. Further, conventional placement does not exploit the knowledge of the distribution of useful LO nodes. It simply uses a predefined function of the address bits to determine the cache index. Even an XOR based placement scheme uses a predetermined function to form the index and makes no further attempt to achieve a more uniform distribution.

A better mapping scheme would be one that can exploit the knowledge of the distribution of useful nodes and yet be flexible enough to adapt to changes in the routing table. In addition, the flexibility achieved in placement should neither significantly increase the hardware complexity, nor increase the hit-latency. We achieve these goals through a two-level mapping framework that adds a few decision bits per IP to decide the RP to which it maps. The remapping from IP to RP ensures better distribution; the usage of different decision bits for each IP makes the remapping adaptive and the placement flexible; and, as the index is identified by using a few additional multiplexers (as we will see in Section 4), the hit time can still be accommodated in one clock cycle.

3.2 Two-level mapping based placement

3.2.1 FLEX

In FLEX, we use 2 conventional bits (b_0, b_1) to first identify the IPs. In Table 4 the last two rows show the number of nodes mapping to each set when each of b_2, b_3 .... b_6 are used in conjunction with b_1 and b_0. It can be seen that using the same final bit for all IPs is not ideal. Instead in FLEX one among {b_2, b_3, b_4, b_5} is chosen to reorganize each IP into two RPs. Note that this bit is chosen on a per IP basis. The chosen bit and number of nodes for each RP are shown in boldface in the last two rows of Table 4. This mapping is better than that achieved by conventional mapping or BS.

3.2.2 IMAP

Even-though the mapping achieved by FLEX is ideal for IP_0, IP_2 and IP_3, it can be seen from Table 4 that for IP_1 we are unable to find a bit which would split the IP into two RPs with equal number of nodes. A closer look at IP_1 (refer Table 3.2.2) reveals that if we further partition it based on bits b_2b_3 and combine the partition $b_2b_3 = 00$ with the partition $b_2b_3 = 11$ to form an RP, we get an uniform mapping for IP_1. This is what we achieve with IMAP. The 4 partitions of Table 4 are further partitioned based on bits b_2 and b_3 to get 16 partitions (refer to Table 5). Now, in each column the IP with the most nodes in it is combined with the IP having the least nodes in it (both shown in bold) to form one RP, and the remaining IPs in the column form the other RP. Note that IP_2 which relies on bit b_4 to get uniformly remapped in FLEX can still be uniformly remapped in IMAP using $b_2 \oplus b_3$. For the example in consideration, using IMAP we obtain eight RPs with 5, 5, 3, 4, 4, 4, 3, and 4 nodes in them.

IP_0	IP_1	IP_2	IP_3
b_6b_5.....b_1b_0	b_6b_5.....b_1b_0	b_6b_5.....b_1b_0	b_6b_5.....b_1b_0
0 **1** 0 0 0 0 0	0 **1** 0 0 0 0 **1**	**1 1 1** 0 0 **1** 0	**1 1 1** 0 0 **1 1**
1 0 0 0 0 0 0	0 **1 1** 0 0 0 **1**	0 **1** 0 0 0 **1** 0	**1 1 1 1** 0 **1 1**
1 1 1 0 0 0 0	**1** 0 0 0 0 0 **1**	**1** 0 **1** 0 0 **1** 0	0 **1** 0 0 **1 1 1**
1 0 0 **1** 0 0 0	**1 1 1** 0 0 0 **1**	**1 1 1 1** 0 **1** 0	**1** 0 0 0 **1 1 1**
1 1 1 1 0 0 0	**1** 0 0 **1** 0 0 **1**	0 **1** 0 0 **1 1** 0	**1 1 1** 0 **1 1 1**
1 1 0 **1 1** 0 0	**1 1 1 1** 0 0 **1**	**1** 0 0 0 **1 1** 0	0 0 0 **1 1 1 1**
0 **1** 0 0 **1** 0 0	0 **1** 0 0 **1** 0 **1**	-	0 **1 1 1 1 1 1**
1 0 0 0 **1** 0 0	**1** 0 0 0 **1** 0 **1**	-	**1 1** 0 **1 1 1 1**
1 1 1 0 **1** 0 0	**1 1** 0 **1 1** 0 **1**	-	-
2 3 6 6 **5**	3 3 6 6 **6**	2 2 **3** 5 4	3 2 4 **4** 2
7 6 3 3 **4**	6 6 3 3 **3**	4 4 **3** 1 2	5 6 4 **4** 6

Table 4: Initial Partitions formed using bits b_0, b_1

		b_1b_0			
		00	01	10	11
b_3b_2	00	3	**4**	**3**	**1**
	01	**3**	2	2	**3**
	10	2	2	1	1
	11	**1**	**1**	**0**	3

Table 5: Number of nodes per partition for 16 IPs

In Table 6 we compare the node to set (IP for one-level map-

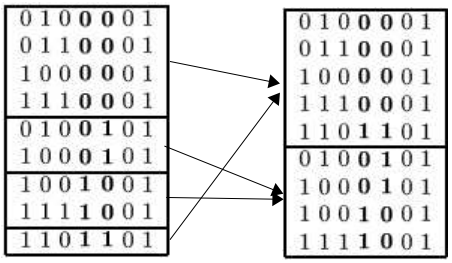


Figure 3: Remapping of IP_1 to RPs using IMAP

Number of Nodes per set	Number of sets			
	base	BS	FLEX	IMAP
1	0	0	0	0
2	2	0	0	0
3	1	4	3	2
4	2	2	3	4
5	1	0	1	2
6	2	2	1	0
7	0	0	0	0

Table 6: Number of partitions having the given number of nodes

ping and RP for two-level mapping) distribution obtained by the various schemes for the routing table in consideration. It can be observed that the IMAP and FLEX schemes achieve a more uniform a distribution of nodes than the base mapping and BS. In Section 6 we quantitatively compare the distributions achieved by the various schemes on real routers.

3.2.3 V-ASSOC

Recent research on caches in the general purpose domain propose ways of achieving flexible associativity so as to make better use of the cache lines [8][15]. However, these proposals are suited only to L2 and L3 caches. Through V-ASSOC we propose a two-level mapping based variable associativity cache that chooses a cache set with a higher (lower) associativity for RPs with more (fewer) nodes. More specifically, the larger two IPs per column of Table 7 (shown in bold) are mapped to larger RPs corresponding to cache sets with more number of lines (higher associativity) in them, while the remaining two are mapped to smaller RPs corresponding to sets with lower associativity. This scheme attempts to distribute the percentage of each partition that can be cached more uniformly.

		b_0	
		0	1
$b_2\ b_1$	00	**5**	**6**
	01	**4**	2
	10	4	3
	01	2	**6**

Table 7: Number of nodes per partition for 8 IPs

4. CACHE ORGANIZATION

In this section we describe the cache organization needed to implement each of the two-level mapping based schemes. All schemes make use of a small number of decision bits to add the required flexibility. The computation of these bits is simple and can be done off-line without affecting the performance of the router. Moreover, as these schemes add only a few additional multiplexers in the implementation they increase the hit latency only by a small fraction. This increase can easily be accommodated within the same clock cycle especially at the low clock rates at which NPs operate. The working of all the schemes primarily involves the identification of two (four for V-ASSOC) candidate sets based on conventional bits, and then choosing one among them based on the decision bits. As the tag match, miss handling and replacement mechanisms are implemented in a manner identical to a conventional cache, in the rest of the section we will focus on the identification of the set to which an address maps (placement). Note that the schemes are applicable to both direct-mapped and set-associative caches of any size. Below we describe the placement in a cache of 1024 sets needing 10 index bits.

4.1 Cache Architecture for FLEX

As illustrated in the previous section, FLEX can flexibly choose its final index bit from among 4 higher order bits, this information is stored using two decision bits per IP. The set to which an address maps is identified as below (refer to Figure 4). The lower order 9 bits are used to identify the IP to which the address belongs and pick out two candidate sets (set0, set1). One among b_{10}, b_{11}, b_{12} and b_{13} is then chosen by the decision bits as the final bit of the index. The selected bit is used to identify between the 2 candidate sets (RPs).

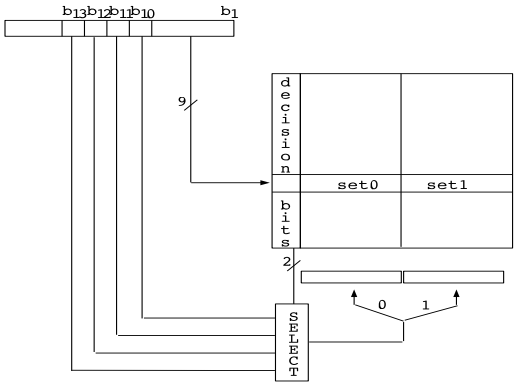


Figure 4: Cache architecture for FLEX

The scheme requires two additional multiplexers, one for selecting the final index bit and the other for choosing one of the two sets. Apart from this $2*512 = 1024$ extra bits are needed to store the decision bits. Whenever the trie used for forwarding is changed and the cache is flushed, the decision bits need to be re-computed and stored. The re-computation takes less than 30 milliseconds real time (measured using the linux time command) on a 2GHz Pentium 4 processor. This can easily be done off-line without affecting performance.

4.2 Cache Architecture for IMAP

As explained previously, IMAP combines the best and worst partition within a *group*[2] of 4 partitions. In IMAP we start with double the number of IPs (2048) and divide them into *groups* such that partitions within a group differ only in bits b_{10} and b_{11}. The number of ways of combining four

[2]the columns of the previous Section (Table 5) are being referred to by a more formal term *groups* here.

partitions of a group into 2 sets (RPs) of two partitions each is given by

$$N_{IMAP} = \frac{4!}{2!2!2!} = 3 \quad (1)$$

These three possible combinations of b_{10} and b_{11} are depicted in Figure 5. It can be seen that for Case 1 of Figure 5, b_{10} identifies the mapping from partitions to sets. Similarly for Case 2 and Case 3, bit b_{11} and $b_{10} \oplus b_{11}$ respectively, will identify the mapping. For a given address bits b_1 to b_9

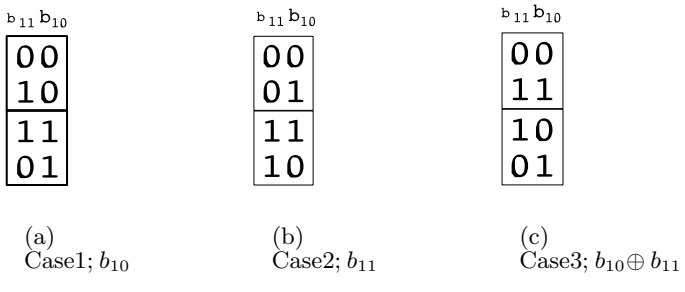


Figure 5: Possible combinations of bits b_{10}, b_{11} for a set

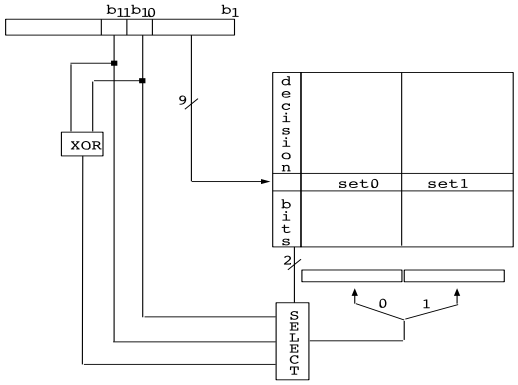


Figure 6: Cache architecture for IMAP

can be used to index into the appropriate group (4 IPs) and pick out the two candidate sets (RPs). For each group the Case to which it conforms is indicated by the means of the 2 decision bits. The corresponding final index bit is chosen based on the Case and used to identify the set among the candidates. As in the case of FLEX two additional multiplexers are required. In addition to this one XOR gate needs to be accommodated. IMAP also requires $2 * 512 = 1024$ bits to store the decision bits.

4.3 Cache Architecture for V-ASSOC

The lower order 10 bits are used to partition the nodes into 1024 IPs. As in IMAP, V-ASSOC then divides the partitions into 4 groups such that the IPs in a group differ only in bits b_9 and b_{10}. The larger two IPs of the group are identified as two larger RPs and mapped into sets with higher associativity and the remaining two partitions are identified as two smaller RPs and mapped to sets with lower associativity. Hence, like in IMAP, the four partitions in a group need to be split into 2 pairs. However, unlike in IMAP, we need to be able to differentiate between the two pairs to indicate which pair gets higher associativity. Therefore, the

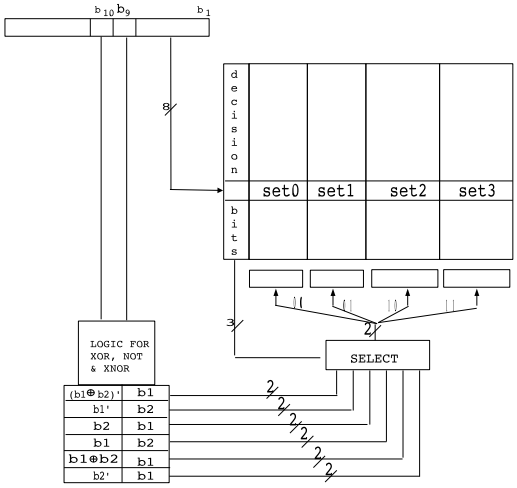


Figure 7: Cache architecture for V-ASSOC

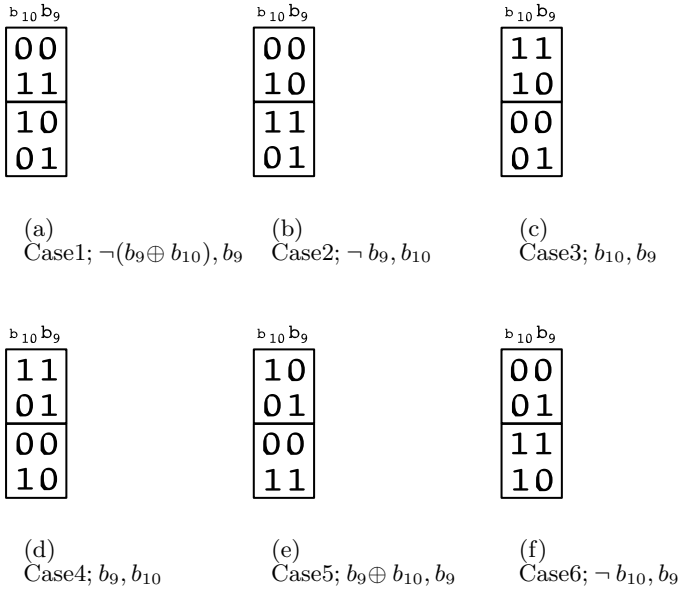


Figure 8: Possible combinations of bits b_9, b_{10} for a pair.

number of ways of dividing four partitions of a group into 2 non-identical sets of two partitions (IPs) each is given by

$$N_{V-ASSOC} = \frac{4!}{2!2!} = 6 \quad (2)$$

These six Cases are depicted in Figure 8. To distinguish between the higher and lower associativity pair we can use $\neg(b_9 \oplus b_{10})$ for Case 1, $\neg b_9$ for Case 2, b_{10} for Case 3, b_9 for Case 4, $b_9 \oplus b_{10}$ for Case 5, and $\neg b_{10}$ for Case 6. Once the pair to which an address maps is identified one among the two sets (corresponding to the two entries in a pair) needs to be identified. For this we can use b_9, b_{10}, b_9, b_{10}, b_9 and b_9 respectively for Cases 1 to 6.

The decision bits of V-ASSOC need to pick one of the six 2 bit pairs of Figure 8. The lower order bits b_8 to b_1 are used to identify the group and pick out 4 candidate sets. Two additional bits are then chosen using 3 decision bits to identify one among the 4 candidates. We therefore need 3

decision bits per group amounting to $3*256 = 768$ additional bits. This scheme also needs two additional multiplexers.

5. EXPERIMENTAL METHODOLOGY

We evaluate the performance of the two-level mapping based schemes using a trace-driven simulation methodology. We use the dineroIV cache simulator [6] (appropriately modified) to measure the number of cache misses. We use two routing tables for evaluation, the FUNET routing table used commonly in literature [14][16], and a recent routing table from the Oregon core router made available through the Route Views Project [21]. The FUNET table contains 41578 entries and its LC-trie representation leads to 128865 trie nodes. The Oregon router contains 161516 entries and 338193 LC-trie nodes. The traces used for simulation and the methodology used to generate them are explained in Section 5.1. The evaluation of speedup in memory access time from cache miss rates is described in Section 5.2.

5.1 Trace Generation Mechanism

Comprehensive evaluation of routers is made difficult by the lack of publicly available IP header/address traces that correspond to publicly available routing tables. Also, the IP addresses in most of these traces are anonymized, and hence, would lose some of the characteristics like prefix length distribution (elaborated below) of the original trace. We overcome these difficulties by using synthetic traces generated using the methodology proposed in [16] (henceforth referred to as *attribute preserving traces*). Attribute preserving traces emulate the *locality* and the *prefix length distribution* characteristics of real traces better than other synthetic traces, like randNET and randIP traces, used in literature [1][11]. We give a brief summary of the method here, more details can be found in [16].

A major disadvantage of randNET and randIP is that they do not preserve the temporal locality characteristics of real traces. Attribute preserving traces introduce locality into the trace by making use of the LRU stack algorithm originally proposed for general purpose applications [19]. The advantage of this algorithm is that the locality in the generated trace can be improved (reduced) by increasing (decreasing) the value of the parameter θ while maintaining the parameter A[3] constant [19].

Another advantage of attribute preserving traces is that they conform to the *prefix length distribution* of realistic traces [13]. It is observed that for real traces, the log of the *average number of addresses that hit a prefix of a given length* decreases linearly with increasing *prefix length*, for prefix lengths ranging from 13 to 24. In particular, in this range, as the *prefix length* increases by one, the log of the *mean number of hits per prefix* decreases by 0.69. Attribute preserving traces emulate this by ensuring that everytime a new entry is added to the LRU stack it is done so that the *prefix length distribution* of the synthetic trace follows that of realistic traces. The traces used for performance comparison are listed in Table 8. Traces pld1M, pld10M and pld100M were generated using the methodology described above. Trace pld1B was generated without using the LRU stack but conforming to prefix length distribution characteristic of realistic traces. The suffix in the trace name gives us a measure of the number of unique entires that would be in the trace if it were to contain a total of 1 billion entries. The total number of addresses in each trace is 1B (with the exception of pld100M) which is 2 to 3 order magnitudes higher than used in earlier studies [12][13][1]. Traces pld1M and pld10M which have more locality represent traffic seen by edge routers. While pld100M and pld1B which have lower locality are representative of traffic seen by core routers [16].

[3]The significance of parameters A and θ is explained in [19].

Name	A	θ	Total addresses	
			FUNET router	Oregon router
pld1M	10	1.8	1B	1B
pld10M	10	1.5	1B	1B
pld100M	10	1.2857	275M	150M
pld1B	-	-	1B	1B

Table 8: Traces used for performance evaluation

5.2 Measuring Speedup in Memory Access Time

The estimation of packet processing throughput and the impact of two-level mapping on it is beyond the scope of this paper. However, it is observed that the use of a multithreaded multiprocessor architecture ensures that there are sufficient resources to feed the memory system continuously with access requests. Further, it has been observed that increasing number of threads gives only diminishing returns and processor utilization saturates at about 32 threads [12] as memory bandwidth becomes the limiting factor. Typically NPs have 64 or more threads and in this scenario the memory access time has a significant impact on packet processing throughput. The average memory access time [9] for a cache inclusive memory hierarchy can be measured as

$$access\ time = hit\ time\ +\ miss\ rate\ *\ miss\ penalty \quad (3)$$

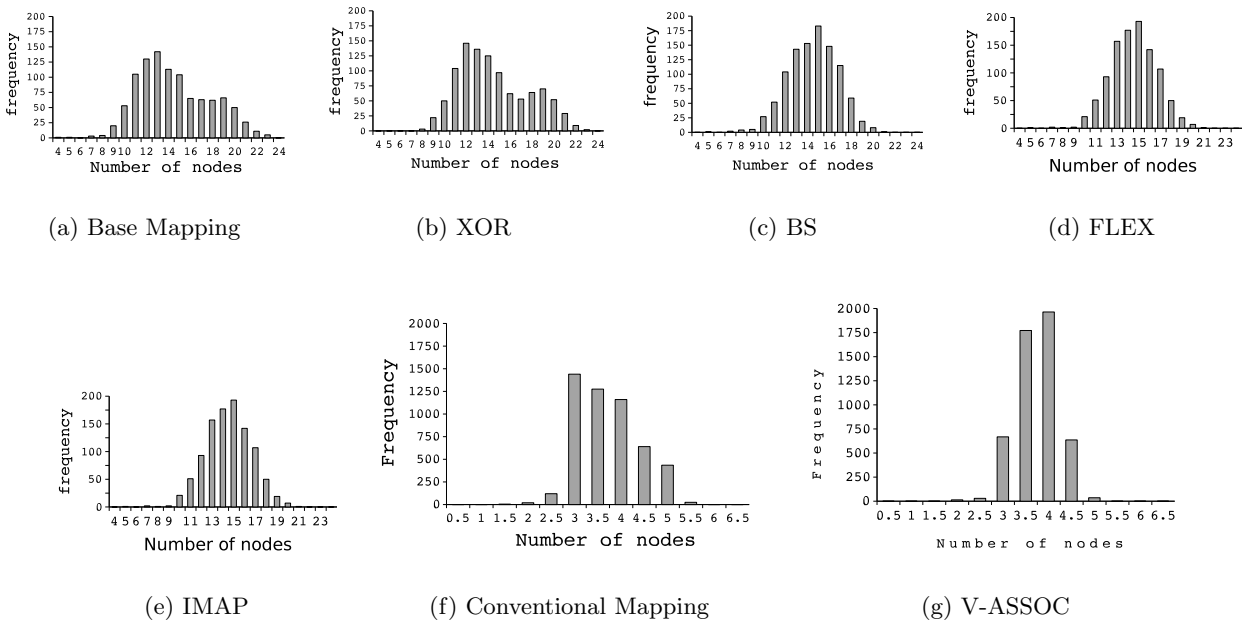
As the information in the auxiliary tables can be stored internally within the LC-trie itself [17] [16], the only memory accesses are trie node accesses. Further, with the use of a segmented cache architecture, accesses to the LO and LL caches can be serviced in parallel. As the LL cache incurs only a small fraction of accesses (reported to be less than 17% [16]) incurred by the LO cache, it is reasonable to assume that the memory access time for the LL cache would be covered by the LO cache accesses. The overall memory access time can therefore be approximated by the memory access time of the LO cache. We can hence calculate the speedup in memory access time using the following formula.

$$speedup = \frac{ht + mr_{base} * mp}{ht + mr_{new} * mp} \quad (4)$$

where mr_{base} is the miss rate of the base LO cache and mr_{new} is the miss rate of the scheme in consideration. We assume a hit time (ht) of 1 clock cycle and a miss penalty (mp) of 100 clock cycles [1][12].

6. RESULTS AND INFERENCES

We compare the various two-level mapping based cache placement schemes with existing one-level mapping based cache placement functions in terms of, (i) the distribution of the useful trie nodes among the cache sets, (ii) reduction in number of misses, and (iii) memory access time speedup. Although we evaluated the performance of caches of various sizes and associativities, due to space constraints we only report the performance for a cache with 8k lines for associativities 1, 2, and 4. For V-ASSOC we simulate a 5k line



(a) Base Mapping (b) XOR (c) BS (d) FLEX

(e) IMAP (f) Conventional Mapping (g) V-ASSOC

Figure 9: Node distribution. y-axis represents number of cache sets in (a)-(e) and cache lines in (f)-(g). In (f) & (g) the x axis values represent ranges not absolute values (5 refers to the range $4.5 < x \leq 5$)

base cache with 5-way associativity and compare it with a V-ASSOC cache of the same size with half its sets having associativity 4 and the remaining half with associativity 6. This ratio of *low* : *high* is chosen based on the observation that the number of nodes mapping to sets with lower associativity is roughly two-third the number of nodes mapping to sets with higher associativity.

6.1 Node to Set Distribution

Figure 9 shows the distribution of nodes to cache sets achieved by conventional mapping, BS, XOR placement, FLEX, IMAP and V-ASSOC for a cache with 1024 sets. These plots are for the FUNET routing table. The trends observed with the Oregon routing table are similar.

In the case of V-ASSOC, each set has either a *high* or a *low* associativity, hence, instead of plotting the number of nodes mapping to a cache set, we plot the average number of nodes mapping to a cache line. As all lines within a set are equivalent, a fair measure for the number of nodes mapping to a cache line is obtained by dividing the number of nodes mapping to a set by the associativity of the set. Intuitively, the scheme with a distribution having a narrow spread around the mean value would be best suited for caching. The natural metric to measure this spread would be the *coefficient of variation* (CoV) of the number of nodes in a cache set (cache line for V-ASSOC). The mean (μ) and coefficient of variation CoV are calculated as

$$\mu = \frac{number\_of\_useful\_nodes}{N} \tag{5}$$

$$CoV = \frac{1}{\mu}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{6}$$

where N is the number of cache sets (lines for V-ASSOC) and x_i is number nodes mapping to the i^{th} set (line).

The CoV values of the various schemes for the FUNET table are shown in Table 9.

Scheme	Useful nodes	N	μ	CoV
Base LO	14859	1024	14.51	0.2251
XOR	14859	1024	14.51	0.2204
BS	14859	1024	14.51	0.1534
FLEX	14859	1024	14.51	0.1464
IMAP	14859	1024	14.51	0.1375
V-ASSOC	14859	5120	2.90	0.1536

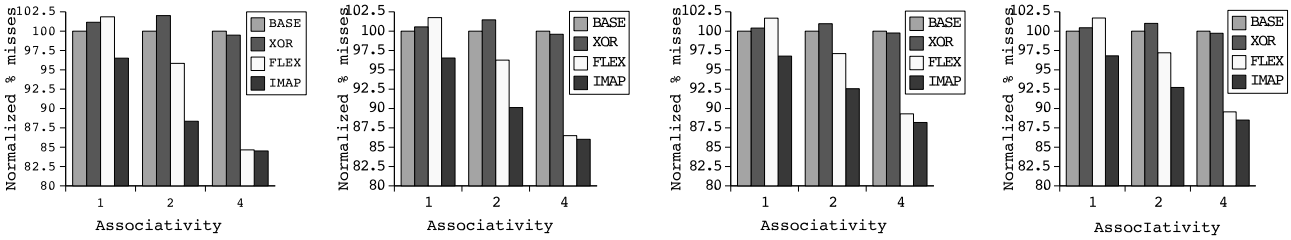
Table 9: CoV values for various schemes

It can be seen that the XOR-based placement scheme, as it does not make use of the knowledge of the distribution of useful nodes, is unable to improve the node to set distribution. In comparison the remaining schemes achieve a lower CoV. While this metric helps rank the schemes, only a detailed cache simulation can tell whether they improve performance.

6.2 Performance of IMAP and FLEX

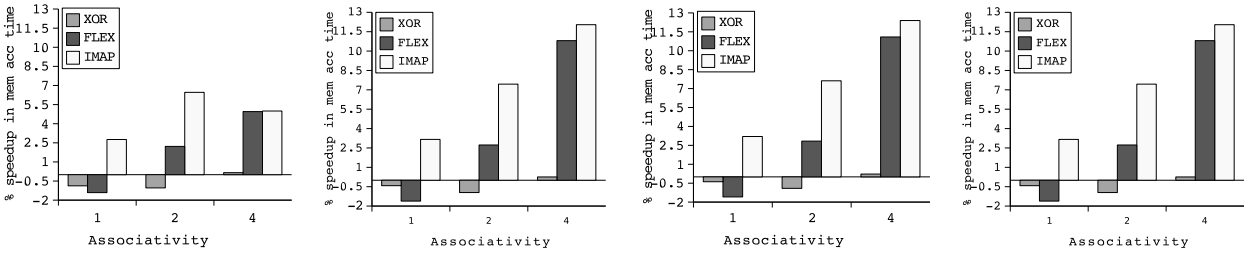
Figures 10(12) and 11(13) compare the miss rate reduction and memory access speedup respectively for the Oregon(FUNET) routing table. The plots in Figure 10 and 12 show the number of misses incurred by various schemes as a percentage of the misses incurred by the base LO cache. The performance improvements for the BS scheme are insignificant and hence are not plotted[4]. Also the performance trends for BS are identical to those of the XOR-based placement scheme. It is observed that, in comparison to the base scheme, the XOR-based scheme performs slightly better for the FUNET route table, while for the Oregon route table it degrades the performance. Even for configurations

[4]This is consistent with the previous results reported for the BS scheme [16].



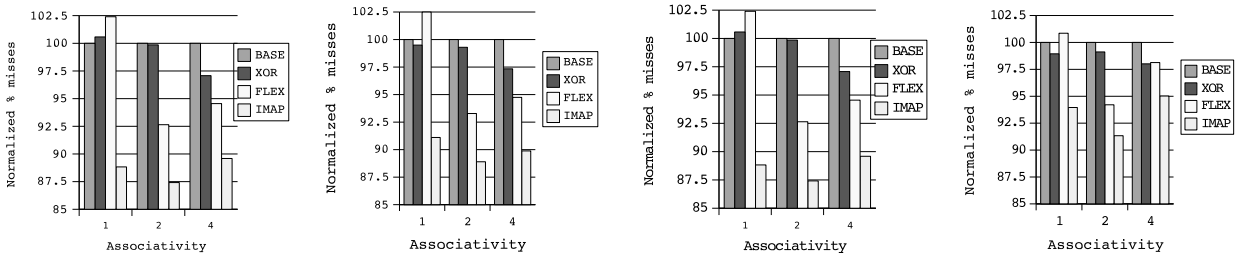
(a) pld1M (b) pld10M (c) pld100M (d) pld1B

Figure 10: Miss rate comparisons for the Oregon route table.



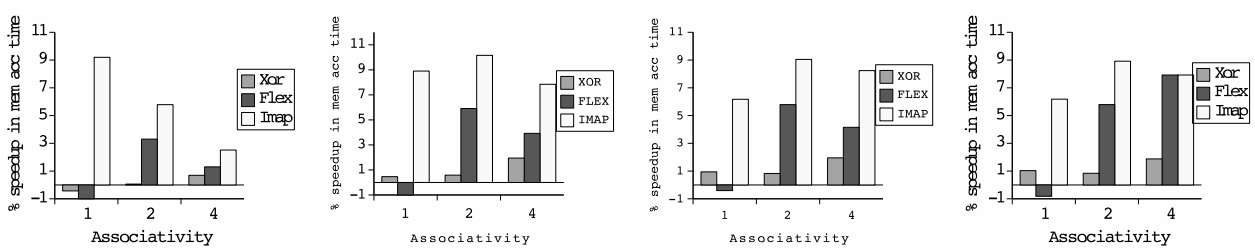
(a) pld1M (b) pld10M (c) pld100M (d) pld1B

Figure 11: Speedup in memory access time, Oregon route table.



(a) pld1M (b) pld10M (c) pld100M (d) pld1B

Figure 12: Miss rate comparisons for the FUNET route table.



(a) pld1M (b) pld10M (c) pld100M (d) pld1B

Figure 13: Speedup in memory access time, FUNET route table.

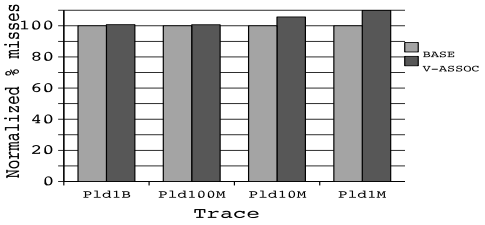


Figure 14: Miss rate comparison for V-ASSOC

where XOR performs better, it at best reduces the number of misses by 3% and leads to a speedup of at-most 2%.
It is seen that for most configurations both FLEX and IMAP perform better than the base cache. Among the two IMAP performs better for all configurations reducing the misses by upto 16% and attaining a speedup in memory acess time of upto 13%. Note that the trace for which we achieve the best miss reduction (pld1M trace) is not the one to attain the best access speedup. This is because of the lower miss rates of the pld1M trace, the trace with maximum locality. This lower miss rate implies that in Equation 4 the hit time (ht) becomes more significant, thereby diminishing the contribution of miss rate reduction towards speedup. It can also be observed that, even though the improvement in miss rates fluctuates with change in locality, the memory access time speedups achieved are more or less consistent.
Finally, it is to be noted that it is unlikely that in any scenario either FLEX or IMAP will degrade performance drastically. This is because, if the conventional bits are able to achieve a uniform mapping, then both schemes will revert to using the same index bits as the base case. That is, both FLEX and IMAP have the choice of picking the base set of index bits if they find nothing better.

6.3 Performance of V-ASSOC

Comparison of V-ASSOC with the base cache (refer to Figure 6.3) shows that V-ASSOC degrades the performance for all traces. It is interesting to note that the *coefficient of variation* of V-ASSOC is slightly higher than that for BS (refer to table 9). As BS does not provide significant benefits, it is not surprising that V-ASSOC doesn't either. Further it is also possible that, even-though in the average case the nodes in a group have a ratio $2:3$ between the two sparser IPs and the two denser IPs, there are still groups for which this does not hold. As V-ASSOC can never revert to the conventional configuration it would induce more conflicts for sets corresponding to such groups and degrade cache performance.

7. CONCLUSIONS

Traditional caches enforce a rigid mapping between the partitions formed by the conventional lower order address bits and cache sets. Such a mapping is unable to distribute the useful trie nodes uniformly and results in increased conflict misses. This paper introduces a two-level mapping framework which exploits the knowledge of the distribution of useful LO nodes to partition the nodes more uniformly among cache sets. Three two-level mapping based schemes for remapping the nodes from naive Initial Partitions to Refined Partitions are proposed. In all of the schemes each IP is individually mapped to one or more RPs with the aid of a few decision bits. We show that all these schemes achieve a better mapping. Performance evaluation indicates that we achieve upto 16% reduction in number of misses and 13% speedup in access time.
In future we plan on adapting two-level mapping to general purpose processor caches and reducing the conflict misses incurred by performance critical memory instructions. We plan to make use of the repetitive phase behaviour of programs to identify an efficient address to set mapping at runtime and use this placement for future accesses.

Acknowledgments

This work was partly supported by a research grant from the Consortium for Embedded and Internetworking Technologies (CEINT) and Arizona State University, Tempe, USA. The authors are thankful to the members of the High Performance Computing Laboratory for their useful comments and discussions.

8. REFERENCES

[1] J-L. Baer, D. Low, P. Crowley and N. Sidhwaney. Memory hierarchy design for a multiprocessor look-up engine. In *Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques*, 2003.

[2] A. Brodnik, S. Carlsson, M. Degermark, and S. Pink. Small forwarding tables for fast routing lookups.In *Proc. of ACM SIGCOMM'97*, 1997.

[3] H.J.Chao. Next Generation Routers. *Invited paper, IEEE Proceeding, vol. 90, no. 9, pp. 1518-1558*, 2002.

[4] T. Chieueh and K. Gopalan. Improving Route Lookup performance using network processor cache. In *IEEE/ACM SC2002 Conf.*, 2002.

[5] T. Chieueh and P. Pradhan. Cache memory design for network processors. In *Proc.of Int. Symp. on High Performance Computer Architecture*, 2000.

[6] J. Edler and M.D. Hill. Dinero IV trace-driven uniprocessor cache simulator. www.cs.wisc.edu/ markhill/DineroIV/, 1998.

[7] A. González, M. Valero, N. Topham, and J.M. Parcerisa. Eliminating cache conflict misses through XOR-based placement functions. *Int. Conf. on Supercomputing*, 1997.

[8] E.G.Hallnor and S.K.Reinhardt. A fully associative software-managed cache design. In *Proc. of Int. Symp. Computer Architecture 2000*, 2000.

[9] J.L. Hennessey and D.A. Patterson. *Computer architecture: a quantitative approach-3rd Ed.* Morgan Kaufmann Publishers Inc., San Francisco, CA, 2003.

[10] M.Kharbutli, K.Irwin, Y.Solilin, J.Lee. Using prime numbers for for cache indexing to eliminate conflict misses. In *Proc. of IEEE Int. Symp. on High Performance Computer Architecture*, 2004.

[11] B.Lampson, V.Srinivasan, and G.Varghese. IP lookup using multiway and multicolumn search. *Proc. of IEEE Infocom*, 98

[12] J. Mudigonda, H.M. Vin and R. Yavatkar. Overcoming the memory wall in packet processing: hammers or ladders? .In *Proc. of Int. Symp. on Architectures for Networking and Communication Systems*, 2005.

[13] G. Narlikar and F. Zane. Performance modeling for fast IP lookups. In *Proc. of ACM SIGMETRICS*, 2001

[14] S.Nilsson and G.Karlsson. IP-address lookup using LC-tries. *IEEE Journal on Selected Areas in Communications*, 1999.

[15] M.K. Qureshi, D. Thompson, Y.N. Patt. The V-Way Cache: Demand Based Associativity via Global Replacement. In *Proc. of Int. Symp. Computer Architecture*, 2005.

[16] K. Rajan and R. Govindarajan. A Heterogeneously Segmented Cache architecture for a packet forwarding engine. In *Int. Conf. on Supercomputing*, 2005.

[17] V.C.Ravikumar, R.Mahapatra, J.C.Liu. Modified LC-trie based efficient routing lookup. *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*, 2002.

[18] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. *IEEE Network Magazine*, 2001.

[19] D. Thiebaut, J.L. Wolf, H.S.Stone. Synthetic traces for trace-driven simulation of cache memories. *IEEE Transactions on Computers*, 1992.

[20] B.Talbot, T.Sherwood, and B.Lin. IP caching for terabit speed routers. *Proc. of IEEE Globcom*, 1999.

[21] University of Oregon Route Views Project www.routeviews.org.