Computer Communications 33 (2010) 390–397

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Expediating IP lookups with reduced power via TBM and SST supernode caching

Ying Zhang [a], Lu Peng [a,*], Wencheng Lu [b], Lide Duan [a], Suresh Rai [a]

[a] Department of Electrical & Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, United States
[b] Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL 32611, United States

ARTICLE INFO

Article history:
Received 17 February 2009
Received in revised form 12 August 2009
Accepted 14 October 2009
Available online 27 October 2009

Keywords:
Tree bitmap (TBM)
Shape shifting trie (SST)
IP lookup
Supernode
Caching

ABSTRACT

In this paper, we propose a novel supernode caching scheme to reduce IP lookup latencies and energy consumption in network processors. In stead of using an expensive TCAM based scheme, we implement a set-ssociative SRAM based cache. We use two different algorithms, tree bitmap (TBM) and shape shifting trie (SST), to organize an IP routing table as a supernode tree composed of a group of supernodes. We add a small supernode cache in-between the processor and the low-level memory containing the IP routing table in a tree structure. The supernode cache stores recently visited supernodes of the longest matched prefixes in the IP routing tree. A supernode hitting in the cache reduces the number of accesses to the low-level memory, leading to a fast IP lookup. According to our simulations, up to 72% memory accesses can be avoided by a 128 KB TBM supernode cache for the selected three trace files, and up to 78% memory accesses can be reduced while using a same size of SST supernode cache. Average TBM and SST supernode cache miss ratios are as low as 4% and 7%, respectively. Compared to a TCAM with the same size, the TBM and SST supernode caches can both reduce 77% of energy consumption.

1. Introduction

Packet routing is a critical function of network processors. An IP router determines the next network hop of incoming IP packets by destination addresses inside the packets. A widely used algorithm for IP lookup is Longest Prefix Matching (LPM). The adoption of a technique Classless Inter-Domain Routing (CIDR) [1] had made address allocation more efficient. In an IP router with CIDR, a ⟨route prefix, prefix length⟩ pair denotes an IP route, where the prefix length is between 1 and 32 bits. For every incoming packet, the router determines the next network hop in two steps: First, a set of routes with prefixes that match the beginning of the incoming packet's IP destination address are identified. Second, the IP route with the longest prefix among this set of routes is selected to route the incoming IP packet.

IP routing table organization and storage is a challenging design problem for routers with increasingly large tables. Many commercial network processors [2,3,4] achieve wire speed IP routing table lookup through high speed memories such as Ternary Content Addressable Memories (TCAMs) and specialized hardware. TCAMs have an additional "don't care" bit for every tag bit. When the "don't care" bit is set the tag bit becomes a wildcard and matches anything. TCAM's fully associative organization makes it parallelly search all the routes simultaneously, leading to low access latency. However, its high cost and high power consumption [5,6] hamper TCAM being widely used.

IP caching has been extensively studied in [7,8,9], where caches are leveraged to provide a fast path for IP lookup to improve the average lookup time. Recently, researchers proposed the replacement of TCAMs by relative less expensive SRAMs. With well organizations, SRAMs can also achieve high throughput and low latency for IP routing table lookup [6,10,11] . In this paper, we propose a supernode based caching scheme to efficiently reduce IP lookup latency in network processors. We utilize two different strategies, tree bitmap and shape shifting trie, to construct two types of supernode trees. Tree bitmap (TBM), which is proposed in [12], organizes an IP routing table to a regular shaped supernode tree. In a 32-level binary tree, we represent it by an 8-level supernode tree if we compress all 4-level subtrees, whose roots are at a level that is a multiple of 4 (level 0, 4,. . ., 28), to be supernodes. On the other hand, shape shifting trie (SST) [13] generates a supernode tree composed of irregular shaped supernodes, which optimized the worst case IP address lookup performance. Lu and Sahni [14] further reduced memory requirement and lookup time for supernode tree. We add a small TBM or SST supernode cache in-between the processor and the low-level memory containing the IP routing table in a tree structure. The supernode cache stores recently visited supernodes of the longest matched prefixes in the IP routing tree. A supernode hitting in the cache reduces the number of accesses to the low level memory, leading to a fast IP lookup. In [15], we demonstrated that a small cache for regular shaped TBMs can significantly reduce the memory accesses and power consumption. In

* Corresponding author.
E-mail addresses: yzhan29@lsu.edu (Y. Zhang), lpeng@lsu.edu (L. Peng), wlu@cise.ufl.edu (W. Lu), lduan1@lsu.edu (L. Duan), srai@lsu.edu (S. Rai).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.10.006

this paper, we further extend the supernode cache to work for not only regular shaped TBMs, but irregular shaped SSTs. We also present more descriptions and experimental results in this version.

While worst case lookup performance is critical to deal with malicious users who attempts to flood network with packets, average lookup performance is another important metric for power consumption and general Internet router throughput. Hence, one category of literature [7,8,9] has been focusing on the average performance metrics. For network processor manufacturers, power dissipation is a first-order concern in processor design. Previous studies [16,17,18] show that persistent high power consumption tends to rapidly heat the processor, and thus largely degrade its reliability and lifetime due to the high temperature, which will introduce serious problems to the processor, such as wear-out of the critical hardware. Reducing average memory accesses is an efficient method to decrease power consumption and improve general Internet router throughput [19,20]. Therefore, we focus on reducing average power consumption and the average number of memory accesses because they can better contribute to a cool network processor.

In our simulation, we compared the TBM and the SST supernode caching scheme with another two caches: a simple set-associative IP address cache and a fully associative TCAM. Several results can be summarized from our experiments: (1) average 69%, up to 72%, of total memory accesses can be avoided by using a small 128 KB tree bitmap supernode cache for the selected three IP trace files, and up to 78% memory accesses can be reduced while using a same size of shape shifting trie supernode cache. (2) A 128 KB of our proposed supernode cache outperforms a same size of set-associative IP address cache 34% in the average number of memory accesses while organizing the IP routing table as a tree bitmap. (3) Compared to a TCAM with the same size, both the TBM and SST supernode cache saves 77% of energy consumption.

The left of this paper is organized as follows. Section 2 introduces related concept of the tree bitmap structure and the shape shifting trie. Section 3 explains the proposed supernode caching scheme. Section 4 lists our experiment results. Section 5 makes a conclusion. Through the paper, we sometimes use the term subtree to indicate a supernode.

2. Related work and background

Many of the data structures developed for the representation of a forwarding table are based on the *binary trie* structure [21]. A binary trie is a binary tree structure in which each node has a data field and two children fields. Branching is done based on the bits in the search key. A left child branch is followed at a node at level i (the root is at level 0) if the ith bit of the search key (the leftmost bit of the search key is bit 0) is 0; otherwise a right child branch is followed. Level i nodes store prefixes whose length is i in their data fields. The node in which a prefix is to be stored is determined by doing a search using that prefix as key.

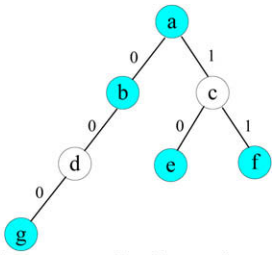
Fig. 1(a) shows a set of 5 prefixes. The * shown at the right end of each prefix is used neither for the branching described above nor in the length computation. So, the length of $P2$ is 1. Fig. 1(b) shows the binary trie corresponding to this set of prefixes. Shaded nodes correspond to prefixes in the rule table and each contains the next hop for the associated prefix. In this paper, we utilize two different optimized trees for the proposed supernode caching.

2.1. Tree bitmap (TBM)

Tree bitmap (TBM) [12] has been proposed to improve the look-up performance of binary tries. In TBM we start with the binary trie for our forwarding table and partition this binary trie into subtries

prefix	value	next hop
P1	*	H1
P2	0*	H2
P3	000*	H3
P4	10*	H4
P5	11*	H5

(a) 5 prefixes

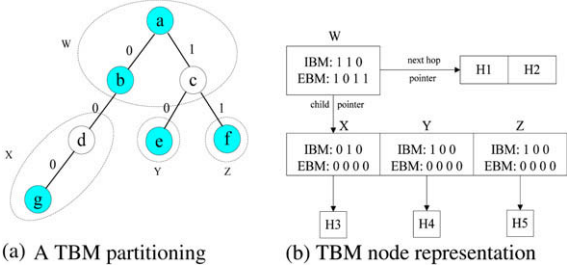


(b) corresponding binary trie

Fig. 1. Prefixes and corresponding binary trie.

that have at most S levels each. Each partition is then represented as a (TBM) supernode. Fig. 2(a) shows a partitioning of the binary trie of Fig. 2(b) into 4 subtries W–Z that have 2 levels each. Although a full binary trie with S = 2 levels has three nodes, X has only 2 nodes and Y and Z have only one node each. Each partition is represented by a supernode (Fig. 2(b)) that has the following components:

1. A $(2^S - 1)$-bit bit map IBM (internal bitmap) that indicates whether each of the up to $2^S - 1$ nodes in the partition contains a prefix. The IBM is constructed by superimposing the partition nodes on a full binary trie that has S levels and traversing the nodes of this full binary trie in level order. For node W, the IBM is 110 indicating that the root and its left child have a prefix and the root's right child is either absent or has no prefix. The IBM for X is 010, which indicates that the left child of the root of X has a prefix and that the right child of the root is either absent or has no prefix (note that the root itself is always present and so a 0 in the leading position of an IBM indicates that the root has no prefix). The IBM's for Y and Z are both 100.
2. A 2^S-bit EBM (external bit map) that corresponds to the 2^S child pointers that the leaves of a full S-level binary trie has. As was the case for the IBM, we superimpose the nodes of the partition on a full binary trie that has S levels. Then we see which of the partition nodes has child pointers emanating from the leaves of the full binary trie. The EBM for W is 1011, which indicates that only the right child of the leftmost leaf of the full binary trie is null. The EBMs for X, Y and Z are 0000 indicating that the nodes of X, Y and Z have no children that are not included in X, Y, and Z, respectively. Each child pointer from a node in one partition to a node in another partition becomes a pointer from a supernode to another supercode. To reduce the space required for these inter-supernode pointers, the children supernodes of a supernode are stored sequentially from left to right so that using the location of the first child and the size of a supernode, we can compute the location of any child supernode.
3. A child pointer that points to the location where the first child supernode is stored.
4. A pointer to a list NH of next hop data for the prefixes in the partition. NH may have up to $2^S - 1$ entries. This list is created by traversing the partition nodes in level order. The NH list for W



(a) A TBM partitioning (b) TBM node representation

Fig. 2. TBM for binary trie of Fig. 1.

is $H1$ and $H2$. The NH list for X is $H3$. While the NH pointer is part of the supernode, the NH list is not. The NH list is conveniently represented as an array.

The NH list (array) of a supernode is stored separately from the supernode itself and is accessed only when the longest matching prefix has been determined. We now wish to determine the next hop associated with this prefix. If we need b bits for a pointer, then a total of $2^{S+1}+2b-1$ bits (plus space for an NH list) are needed for each TBM supernode. Using the IBM, we can determine the longest matching prefix in a supernode; the EBM is used to determine whether we should move next to the first, second, etc. child of the current supernode. If a single memory access is sufficient to retrieve an entire supernode, we can move from one supernode to its child with a single access. The total number of memory accesses to search a supernode trie becomes the number of levels in the supernode trie plus 1 (to access the next hop for the longest matching prefix).

2.2. Shape shifting trie (SST)

For dense binary tries, the TBM algorithm is space efficient. However, for binary tries in which the nodes are distributed sparsely, TBM is not flexible enough to make the best use of the memory space. Song et al. [13] propose an innovative strategy called shape shifting trie (SST) to organize an IP routing table. SST can efficiently use the memory space for sparse binary tries and largely improve the worst case performance of IP lookups. Several methods can be used to construct an SST from a given binary trie; and the one generated by the breadth-first pruning (BFP) algorithm has the minimum height for a given binary trie. In our work, we use the BFP algorithm to construct an SST.

For a given binary trie, we first traverse all the nodes in the breadth-first-order. Each time when we access a node x, we calculate the number of its descendants. Suppose $S(x)$ denotes this number, which also represents the amount of nodes of the subtree rooted at x. We also define a fixed value as the maximum number of nodes that can be contained within an SST supernode, assuming K indicates this bound. If $S(x)$ is less than or equal to K, we assign x and all of its descendants to a new SST supernode and prune these nodes from the original binary trie. We repeat this procedure until all the nodes in the binary trie are assigned. Assume H + 1 passes of BFP execution is needed to classify all the nodes of the given binary trie into SST supernodes, then the SST height is H. Like a TBM supernode, an SST supernode also contains necessary information for IP lookup. For an SST node including K binary nodes, the following components are contained:

1. IBM (internal bitmap). It indicates whether a binary node in a supernode contains a valid prefix. A binary node that contains a prefix is considered to be valid. This bitmap has exactly the same meaning with that of a TBM supernode. An SST supernode with K nodes needs K bits to record its IBM.
2. SBM (shape bitmap). It describes the shape of a supernode. To encode the SBM of a supernode, we consider the supernode as an independent tree and augment it with additional dummy nodes. We then assign a bit to each node in the augmented tree, including the dummy nodes. Specifically, we assign a "1" to an original node and a "0" to a dummy node. For example, in Fig. 3, the SBM of the supernode W, which is composed of a, c, f, is 01 01 00. The first pair (0, 1) is corresponding to node a. It means that the left child of node a is a dummy child and the right child is an actual one. The second pair (0, 1) corresponds to node c which has the same meaning as the first pair. As for the last pair, (0, 0) indicates that both the left child and right child of node f do not belong to this SST supernode. We can see that each binary node in an SST supernode needs a pair of bits to represent its children information, so 2K bits are indispensable to record the SBM of an SST supernode containing K nodes.
3. EBM (external bitmap). It gives information of the potential exit points from a supernode. Here, the meaning of the EBM of an SST supernode is different from that of a TBM supernode. We can get the exact information of the EBM by referring to the SBM of this supernode at the same time. Consider the example in Fig. 3, the EBM of the supernode W is 1011. The first "1" indicates that node b, the left child of node a, is an exit point from this supernode. To get this information, we access the SBM of supernode W. Since the first bit of the SBM is "0", we know that the left child of node a does not belong to W. This means that node b is an exit point from supernode W. Furthermore, we also know that node b is an actual binary node because a "1" in EBM indicates an actual node while a "0" means a dummy node. The second bit of EBM indicates that the left child of c is an exit point which is a dummy node. We can also observe that each 0 in SBM implies a potential exit path in EBM. The last two "1"s imply that node g and h are both actual exit points. We need K + 1 bits to encode the EBM for a K-node SST supernode.
4. A child pointer. It points to the first children SST node of the given supernode.
5. A next hop pointer. It points to the next hop information for the first valid binary node within the SST supernode. In order to use the memory efficiently, the children and the next hop information of a given SST node are stored sequentially. This allows the access to all the SST children supernodes and the next hop information entries by using only the pointer to the first children supernode and hop information entry.

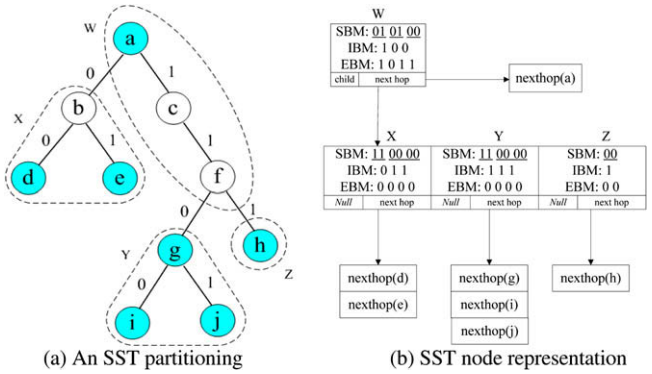


Fig. 3. An SST and its representation.

In Section 2.1, we have calculated that a total of $2^{S+1}+2b-1$ bits are needed for a TBM supernode with a stride of S. Specifically, we use 2^S-1 bits for IBM, 2^S bits for EBM and 2b bits for pointers. However, if we consider the S-stride subtree to be an SST supernode, an additional $2\times(2^S-1)$ bits are required to store the SBM. Therefore, we need $2^{S+2}+2b-3$ bits in all to represent an SST supernode. In this paper, we assume that both TBM and SST supernodes have the same size of memory block. Consequently, an SST supernode containing K binary nodes should satisfy the following condition:

$$2^{S+1}+2b-1=K+K+1+2\times K+2b$$

In practice, different sized padding bits may be used to fit the supernode structure in a memory block. Here, we assume that the padding bits for both structures have the same size. From the above equation, K should be approximately no more than 2^{S-1}. Therefore, if we set the stride of a TBM supernode to be S, the maximum number of binary nodes in an SST supernode will be accordingly set to 2^{S-1}.

3. Supernode caching

We assume that the supernode tree is stored in a low-level memory. According to our analysis in Section 2, the number of memory accesses is largely dependent on the length of the longest prefix match (LPM). Therefore, we introduce a small supernode cache to reduce the number of low-level memory accesses. Similar to a traditional cache in a PC, the proposed supernode cache locates in-between the processor and the low-level memory. Fig. 4 illustrates the overview of our design. A cache hit marked by a dash line will save a number of memory accesses to supernodes. We compress the binary routing table tree into a TBM supernode tree which is stored in the low level memory. If a supernode corresponds to a 4-level subtree, a 32-level binary tree is compressed into an 8-level TBM supernode tree. Assume that each supernode access takes one memory access, the maximum number of memory accesses for an IP lookup is 9: it reads seven supernodes plus the root nodes and searches the next hop for the longest matching prefix. When the root supernode is always held in cache, this number becomes 8. Obviously, maintaining a small cache will help to reduce the number of memory accesses. The situation for SST caching is the same with that of TBM.

Fig. 5 depicts the working procedure of the proposed cache for both TBM and SST. We assume that the TBM or the SST of an IP routing table has been constructed and stored in the low level memory. The cache data array stores pointers to the locations of the corresponding Supernode. For all supernodes whose address is less than or equal to 8 bits, we store them in a separate fully associative cache. This is practical because there are at most $2^9 - 1$=511 addresses. Other supernodes are stored in the cache depicted in the above figure. There is a mask field which is associated with each tag entry and records the number of bits for each tag. For example, a 21-bit tag will have its related mask value 21. The number of mask bits is 5 because the longest tag has 24 valid bits. Note that the number of mask bits can be reduced to be 3 for an 8-level TBM supernode tree because only 5 types of TBM supernodes with length 12, 16, 20, 24 and 28 will be stored into this cache. For an incoming destination IP address, the working procedure of the cache is as follows.

(1) Set identification by the left most 8 bits. This step works as the index selection in a cache. Our design is slightly different from a normal cache in that we use the left most 8 bits of an address as the index for the purpose of the longest prefix match searching,.
(2) For each entry in the selected set, we extractits mask field and convert it to a 24-bit mask and send it to a mask register. The value of the mask field will determine the number of "1" bits in left most. Other bits will be filled as "0" bits. For example, a mask field stores a value of 21, then the converted results has 21 "1" bits in the left most and 3 "0" bits for the remainder bits.
(3) Filter out unused bits in the destination IP address by bit-and operation between the IP address and the mask register, and then compare the result with the tag entry. This can find potential matched supernodes stored in the cache.
(4) After the tag comparison, we select the data entry corresponding to a matched tag with the largest mask value. This ensures the longest IP prefix matches. The data entry stores the address of the supernode in memory. A cache hit can save the number of memory accesses to the supernode. The search continues from the matched supernode in memory and proceeds downwards until the longest matched prefix is found. If the supernode containing the longest matched prefix is found in the cache, we only need one memory access for the next hop.

To better demonstrate the working procedure of our design, we show the search process of an arbitrary IP address, take 192.168.20.11 for example, in a router with the proposed supernode cache. For simplicity, we convert the IP address to binary format as 11000000.10101000.00010100.00001011. Suppose the cache is configured to two-way set-associative. In step one depicted above, 11000000 is chosen as the set index, which corresponds to a specific set in the cache. We assume the two entries in the selected set are already filled by previous cache update. Thus in step two, the convertor will generate two bit streams for each entry individually based on their mask values. Without loss of generality, we assume the mask field values of these two entries are 15 and 20, respectively. As a result, the 8th to the 22nd bits of the destination IP address are selected to compare with the tag of entry one (15 valid bits), while the 8th to 27th bits are filtered out for entry two (20 valid bits). Suppose both of the two groups of selected bits match the tags, then the entry with 20 as its mask value is consequently selected for further process since it means a potential longer matching prefix. Therefore, the address of the corresponding supernode is obtained immediately from the data entry and the lookup will continue by accessing the supernode in the low-level memory. If the search ends in the current supernode, the next hop information is available promptly from the supernode data structure; otherwise, deeper supernodes will be accessed to get potential longest matching prefix in the low level memory.

A cache miss will result in cache update and replacement. We employ an LRU replacement policy: we replace the least-recently-used entry with the longest matched prefix's supernode. Consider the above example, if none of the tag array entries in the selected set (according to the index bits) matches the tag bits for the incoming IP address (192.168.20.11), a cache miss occurs. This will lead to

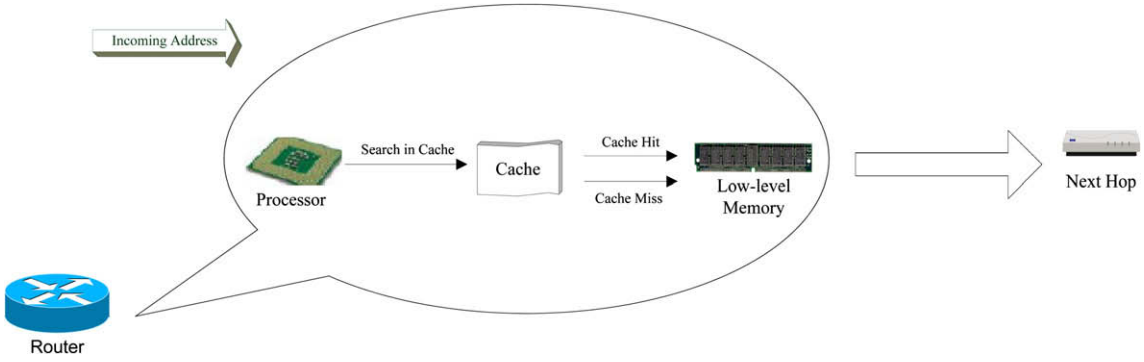


Fig. 4. High-level architecture of the design with supernode cache.

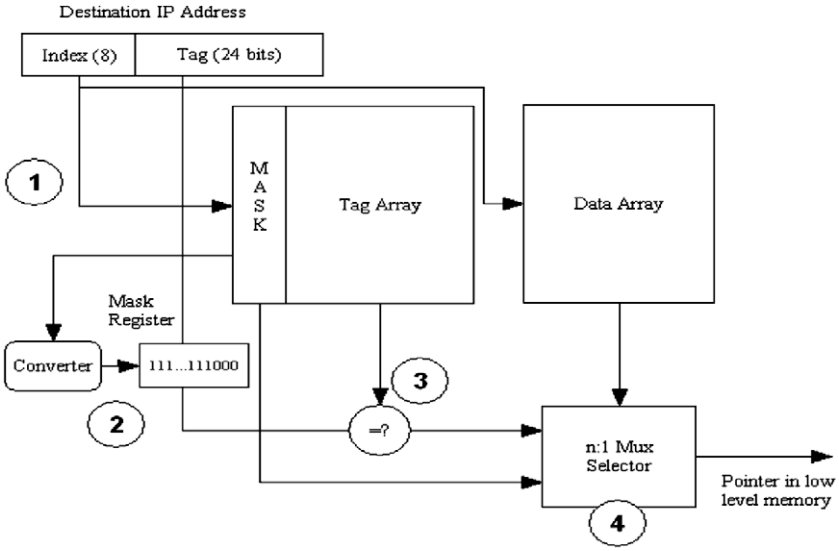


Fig. 5. Design of the supernode cache for both TBM and SST.

the lookup to be restarted in the low-level memory. If the longest matched prefix has 26 bits and the ending supernode has a length of 24 (either for TBM or SST), we update the cache as follows. First, we store bits 9-24 (10101000.00010100) to the tag entry, then we update the mask field with a value $24 - 8 = 16$. Finally, the location of the supernode in the low-level memory will be stored into the corresponding data entry of the cache. Note that this procedure is the same for both TBM and SST cache.

By leveraging the above cache design, we directly jump to a supernode in the search path of the bitmap tree, skipping over its ancestor supernodes along the path. However, we may fail to find the longest matching prefix if it exists in one of these ancestor supernodes. For example, consider the tree bitmap in Fig. 2 and an incoming packet with the destination address 001*. We start the search of cache, if the address of supernode X is found, then the search continues at X and returns no match, though prefix 0* of binary node b should be returned. We use the *covering prefix* strategy as proposed in [22] to solve this problem by pushing to the underlying binary root of each supernode a valid prefix from its lowest ancestor binary node. In this case, 0* of b is pushed down to d, and the search of X will successfully return 0* for the longest match of the destination address 0001*.

4. Experimental results

To evaluate the proposed supernode caching scheme, we download a routing table RS1221 from [23] and download three trace files from routers *ipls*, *svl* and *upcb* in the website [24]. In the following experiments, we collect statistics of first 1.5 million IP addresses whose longest prefix is larger than zero, i.e, matching an inner node in the IP prefix tree. Totally, we implement five schemes with the longest prefix matching algorithm for IP routing: (1) without cache; (2) with an IP address cache; (3) with a TCAM; (4) with a TBM supernode cache; (5) with an SST supernode cache. For each scheme, we count the average memory access time. If there is a cache, we also measure miss ratios. In addition, we simulate energy consumption for each cache scheme.

In our experiments, we assume a 4-bit stride tree bitmap, and set the maximum number of the binary nodes that can be held in an SST supernode to be 8, according to the previous discussion in Section 2.2. In the no cache scheme, we use different methods to calculate the memory access time for TBM and SST respectively, since they are constituted with distinct shaped supernodes. For TBM, assuming that L denotes the number of steps taken to find the longest prefix node, the number of memory access for an IP lookup in the no cache scheme is $L/4 + 1$. For SST, if X supernodes are needed to be visited along the search path of the address, then $X + 1$ memory accesses are necessary for the no cache scheme of SST. In the second scheme, we design an IP address cache which contains the next hop information pointer in each entry. It can easily be implemented as a set-associative cache by selecting part of the IP address bits as the set index and left bits as the tag. If an incoming IP address matches an entry in the IP address cache, it requires only one memory access to obtain the next hop. Otherwise, it needs $L/4 + 1$ memory access. In the third scheme, we assume that there exists a Tenary CAM. If an incoming IP address matches an entry in the TCAM, it requires only one memory access to obtain the next hop. Otherwise, it needs $L/4 + 1$ memory access. In the forth scheme, if the TBM supernode cache hits, it takes $\lceil (P - C)/4 \rceil + 1$ memory accesses. Here P denotes the length of the longest prefix and C denotes the length of TBM supernode prefix hit in the cache. Otherwise, if the supernode cache misses, it requires $L/4 + 1$ memory accesses. In the fifth scheme, suppose that it is necessary to access M supernodes to get the longest prefix in the no cache scheme and we find the Nth supernode is in the cache, then it takes $M - N + 1$ memory accesses to get the longest prefix for a given address. Here the Nth node is either an ancestor of the Mth node or the Mth node itself. In all of our experiments, we don't prefetch the routing tables into cache. Therefore, compulsory misses will be also included as misses.

As stated in Section 1, we focus on reducing average power consumption and the average number of memory accesses because they can better contribute to a cool network processor. Fig. 6

Trace File	IPLS	SVL	UPCB
Date	06/01/2004	03/18/2005	02/19/2004
Avg. Accesses in TBM	4.28	4.73	4.64
Avg. Accesses in SST	6.45	7.16	7.06

Fig. 6. Average memory accesses of 1.5 million IP addresses from each trace file.

shows the average numbers of memory accesses for the three selected trace files. Among the three files, IPLS has the smallest average number of memory accesses while SVL has the largest number. To understand the details, we further collect distributions of longest prefix matching (LPM) in Fig. 7. Two observations can be made from it: (1) most LPM hit the range from prefix length 8 to length 24. There is no matching for prefix length less than 8, therefore we do not show this range, while there are only a very few LPM hit prefixes longer than 24. (2) Three trace files show different distribution. IPLS has the largest group with prefix length 8 while SVL and UPCB have the largest two groups with length 16 and 17. One can expect that the proposed supernode caching scheme will have more performance benefits for SVL and UPCB than for IPLS because more relatively long supernode prefixes can be found from the supernode cache, resulting in less memory accesses.

According to Fig. 6, we find that the average search time of SST are longer than that of TBM though its worst case performance is better than that of TBM. SST can generate a trie with the minimum height, indicating that the maximum number of memory accesses in SST is better than that in other tries. However, the average memory accesses are not necessary superior. To give an example, consider a destination address which exactly matches an entry with the prefix length of 32. In TBM, we have to visit 8 supernodes to get this prefix, but only 6 or 7 supernodes need to be accessed in SST. From the distribution shown in Fig. 7, we know that most LPM has the length of 8, 16 and 17, which implies that most of the memory accesses in TBM are less than 5. However, in SST, the binary nodes which contain valid prefixes with the length of 8, 16 and 17 may locate in the supernodes with depth more than 5, resulting in a larger average number of memory accesses.

To illustrate the effect of supernode cache, we simulate different cache sizes ranging from 8 KB to 128 KB. We assume each cache entry has a four-byte width which can only store one unit because no spatial locality for IP addresses streams [10][25]. We assume that all 8-bit supernodes are stored in a separate small cache. This is reasonable because the maximum size of this additional cache is 256 * 4 = 1 KB for TBM and 512 * 4 = 2 KB for SST. We use the mask field in the cache tag array to find the longest prefix matching. This implementation makes all supernode caches have fixed 256 sets. When the total cache size increases, we increase the set size instead of increasing the number of sets. In the TBM scheme, there are five types of supernodes with different lengths will store in this cache: 12, 16, 20, 24 and 28. In the SST scheme, the supernodes with lengths from 9 to 32 are all possibly stored in the cache. We distinguish the different length of supernodes by the mask bits. To make the comparison consistent, we also design the same total sizes and set sizes of IP address caches.

Fig. 8 illustrates average numbers of memory access with three caching schemes for each trace file. In general, all caching schemes reduce the average number of memory accesses. For SVL, a 32 KB IP address cache, TCAM and TBM supernode cache reduce the average number of memory accesses from 4.73 to 2.11, 2.08 and 1.57, representing 55%, 56% and 67% reduction respectively. In this case, the TBM supernode cache outperforms the IP address cache 34% and the TCAM 32%. The average memory access reductions of the three caching schemes with 32 KB are 50%, 51% and 62% separately for the selected three trace files. On the other hand, for SVL, the SST supernode cache can reduce the average number of memory accesses from 7.16 to 1.97 which represents 72%, and it reduces 69% of the accesses for the three trace files in average. The TBM supernode cache shows the best performance among these caching schemes in all cases. When the cache size reaches 128 KB, the TBM supernode caching scheme's average memory access numbers for the three trace files are 1.51, 1.31 and 1.46, which means that 65%, 72% and 69% memory accesses are reduced. The average memory access reductions of the caching schemes are 52%, 54% and 69%, respectively for the selected three trace files with a 128 KB cache size. The SST supernode cache reduces the accesses to 1.77, 1.54 and 1.65, representing 75%, 78% and 74%.

We also collect cache miss ratio information and present them in Fig. 9. Several observations can be made: (1) the supernode caches, both TBM and SST, have smaller miss ratios than another two

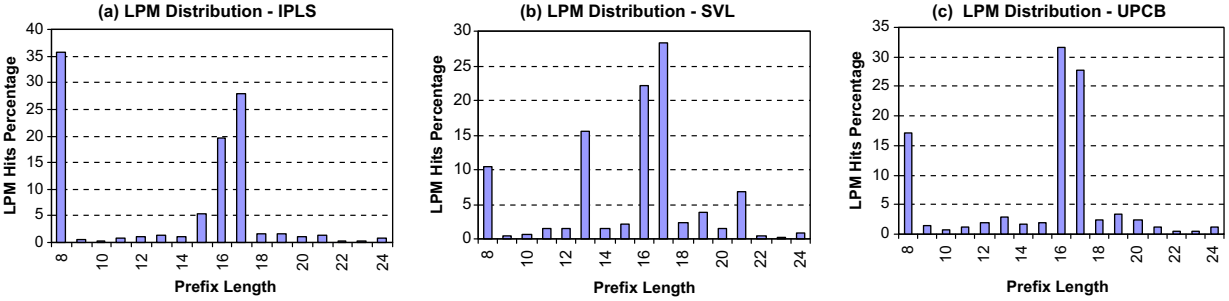


Fig. 7. LPM distributions of the three trace files.

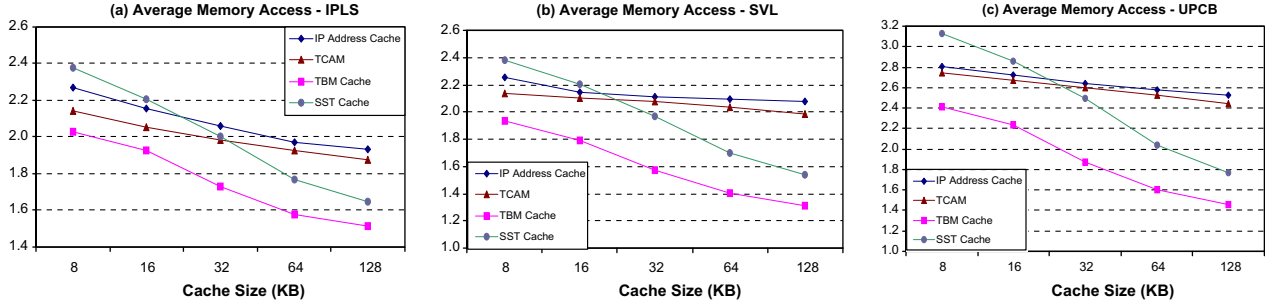


Fig. 8. Average memory access of the three trace files.

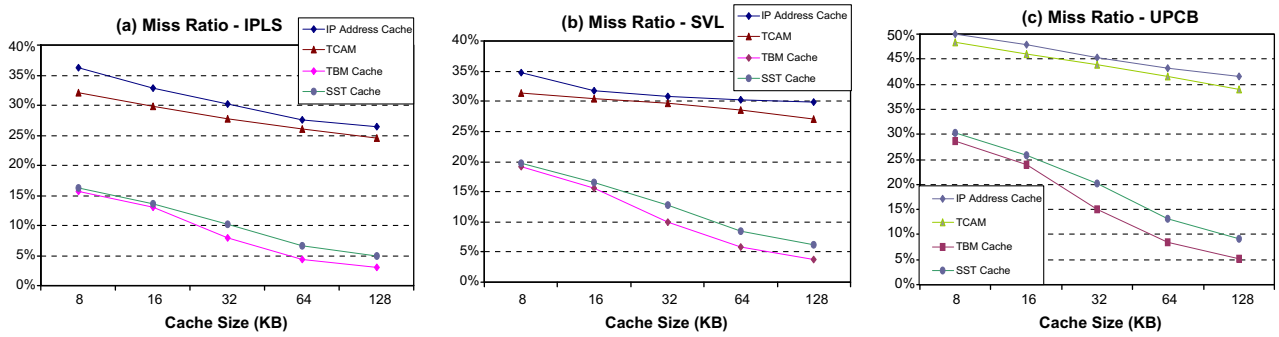


Fig. 9. Miss ratios of three caching schemes for each trace file.

schemes, catching the strongest temporal locality in all cases. This is reasonable because a supernode, representing a subtree, will get reused if any node inside the subtree appears. The IP address cache and the TCAM will hit only if the same IP address recurs. For the three trace files, the miss ratios for a 32 KB TBM supernode cache are 8%, 10% and 15%, respectively, and that for a same sized SST supernode cache are 10%, 13% and 20%. When the cache size reaches 128 KB, the miss ratios for a TBM supernode cache can be further reduced to 3%, 4% and 5%, while that for an SST cache are 5%, 6% and 9% separately. The miss ratio for the SST supernode cache is a bit higher than that of the TBM cache. This is due to the irregular shape of an SST supernode. To demonstrate this, we assume a TBM and an SST constructed from two identical binary tries. One can expect that for an S-stride TBM supernode, the root binary node, denoted by x, and all of its descendants within $S-1$ generations, are contained in this supernode. This implies that an IP address' search path which passes node x and ends at one of the following $S-1$ bits will hit this supernode cache if it exists in the cache. However, this cannot be guaranteed in the SST since any descendant of node x is possible to locate in a different SST supernode, which results in a larger miss rate for the SST cache; (2) the slope of the IP address cache and the TCAM cache miss ratio lines are more flat than that of the TBM supernode cache and SST supernode cache. This means that temporal locality of the IP address streams are limited. The possibility of recurrence of a supernode is larger than that of an IP address. Actually, this is the theory foundation of that a supernode cache outperforms an IP address cache.We also performed a sensitivity study for the SST cache. When the SST supernode size increases, the average search times can be further reduced. Consequently, the memory block size containing the SST supernodes will also be increased. Fig. 10 demonstrates the average search times for the three trace files, with a 128 KB cache but different SST supernode sizes. We can observe that the performance benefits from a large SST supernode almost reach its limit when the size is 24.

To illustrate power efficiency of the proposed supernode cache, we simulate energy consumption of the four caching scheme. We use CACTI 4.1 [26] to simulate the IP address cache and the supernode caches because they are set-associative cache. CACTI is firstly released to help computer architects to model SRAM caches, and now is a tool widely used to measure cache power, estimate cache area and other circuit features. For the power issue which we are interested in this work, CACTI can accurately compute the total dynamic read power/energy, write power/energy. It is sensitive to a group of configuration parameters including cache size, associativity, tag bits length and manufacture technology, and etc. In our simulation, we input different sets of parameters that correspond to each scheme to CACTI and measure the power consumption individually. Specifically, for the caches used in the four schemes, we assume they work under a *Vdd* of 1.7V and 0.18 μm technology. We configure the cache as set-associative and set the block size to be 4 bytes. Cache size is increased from 8 KB to 128 KB. To make comparison fair, we also include power consumption of the small 1 KB fully associative cache which includes all 8-bits supernodes in the TBM supernode cache scheme and the small 2 KB fully associative cache in the SST supernode cache scheme. According to Fig. 7, the average activity of this small cache is about 20% for all three trace files. Therefore, in our simulation, we include 20% power consumption for the fully associative caches. We also simulate TCAM's power consumption using a recent model [27]. Fig. 11 depicts read energy which represents most of energy consumption of four caching schemes. From this figure, we can see that both the TBM and SST supernode cache have a little higher energy consumption than the simple IP address cache because 20% additional searches fall into the small 1 KB or 2 KB fully associative 8-bit supernode cache. For the supernode caches, the energy consump-

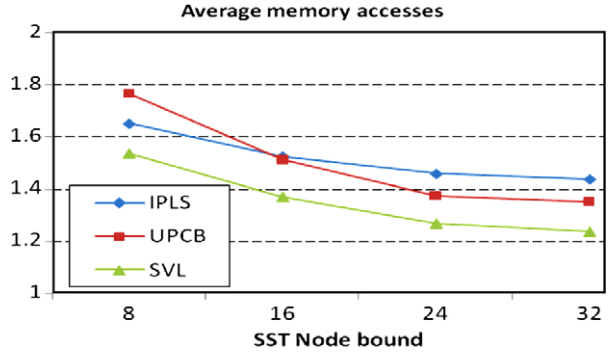


Fig. 10. Average memory accesses in SSTs with different supernode sizes.

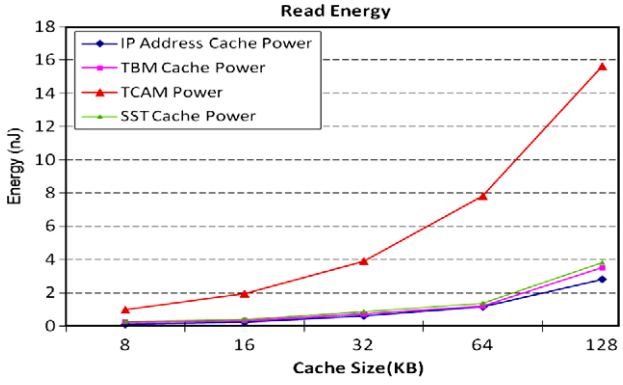


Fig. 11. Read energy comparison of the three caching schemes.

tions are very close to that of the simple IP address cache. The SST cache consumes slightly higher power than the TBM cache does. This is because (1) the SST cache has 2 bits more in its tag array as discussed in Section 3; (2) the fully associative cache has larger size (2 KB). However, the supernode caches still demonstrate significant advantage comparing to the TCAM. For a 128 KB cache, the read energy consumptions for TCAM, SST, TBM are 15.64 nJ, 3.64 nJ and 3.61 nJ, respectively, which means both supernode caches save 77% energy consumption compared with the same size of a TCAM.

5. Conclusion

In this paper, we propose a novel supernode caching scheme to reduce IP lookup latencies and energy consumption in network processors. In stead of using an expensive TCAM based scheme, we implement a set-associative SRAM based caching scheme. The proposed supernode cache can work for multiple organizations of a binary IP routing tree: not only for regular shaped tree bitmaps (TBMs), but also for irregular shaped shifting tries (SSTs). The supernode cache, which stores recently visited supernodes of the longest matched prefixes in the routing table, is placed in-between the processor and the low level memory which contains the IP routing table in a tree structure. A supernode hitting in the cache reduces the number of accesses to low level memory, leading to a fast IP lookup. When the cache size is set to be 128 KB, a TBM supernode cache can save an average 69%, up to 72%, of total memory accesses for the selected three IP trace files, and it ourperforms a same sized set-associative IP address cache 34% in the average number of memory accesses. For SST, which is originally proposed to improve the worst case performance, our supernode caching scheme greatly improve its average performance: an average 76%, up to 78%, of total accesses can be reduced for the three selected trace files. Compared to a TCAM with the same size, the TBM and the SST supernode cache can both save 77% of the energy consumption. The supernode cache works better for a trace file with larger groups LPM hits in relatively long prefixes. Our results also illustrate that the supernode caches catch stronger temporal locality than the other two cache schemes do.

Acknowledgment

This work is supported in part by the Louisiana Board of Regents grants NSF (2006)-Pfund-80, NSF (2009)-Pfund-136 and LEQSF (2006-09)-RD-A-10, the Louisiana State University and an ORAU Ralph E. Powe Junior Faculty Enhancement Award. The authors thank Dr. Sartaj Sahni for his feedbacks on the draft. Anonymous referees provide helpful comments.

References

[1] Y. Rekhter, T. Li, An architecture for IP address allocation with CIDR, RFC 1518 (1993). September.
[2] EZ Chip Network Processors, Available from <http://www.ezchip.com>.
[3] Intel IXP2850 Network Processor, Available from <http://www.intel.com/design/network/products/npfamily/ixp2850.htm>.
[4] Network and Communications ICs, Available from <http://www.agere.com/enterprise_metro_access/network_processors html>.
[5] F. Zane, G. Narlikar, A. Basu, CoolCAMs: Power-Efficient TCAMs for Forwarding Engines, IEEE INFOCOM, April 2003.
[6] S. Kaxrias, G. Keramidas, IPStash: a power-efficient memory architecture for IP-lookup, in: Proceedings of the 36th International Symposium on Microarchitecture, December 2003.
[7] J. Fu, O. Hagsand, G. Karlsson, Improving and analyzing LC-trie performance for IP-address lookup, Journal of Networks 2 (3) (2007).
[8] B. Talbot, T. Sherwood, B. Lin, IP caching for terabit speed routers, Globecom (1999).
[9] T. C. Chiueh , P. Pradhan, High-performance IP routing table lookup using CPU caching , IEEE INFOCOM 1999.
[10] T. Chiueh, P. Pradhan, Cache memory design for network processors, in: Proceedings of the Sixth International Symposium on High Performance Computer Architecture, pp. 409–418, Feb. 2000.
[11] T. Sherwood, G. Varghese B. Calder, A pipelined memory architecture for high throughput network processors, in: Proceedings of the 30th International Symposium on Computer Architecture (ISCA), June 2003.
[12] W. Eatherton, G. Varghese, Z. Dittia, Tree bitmap: hardware/software IP lookups with incremental updates, Computer Communication Review 34 (2) (2004) 97–122.
[13] H. Song, J. Turner, J. Lockwood, Shape shifting tries for faster IP route lookup, in: Proceedings of the13th IEEE International Conference on Network Processors, November 2005.
[14] W. Lu, S. Sahni, Succinct representation of static packet classifiers, IEEE/ACM Transactions on Networking 17 (3) (2009).
[15] L. Peng, W. Lu, L. Duan, Power efficient IP lookup with supernode caching, in: Proceedings of the 50th IEEE Global Communications Conference (Globecom), November 2007.
[16] S.H. Gunther, F. Binns, D.M. Carmean, J.C. Hall, Managing the impact of increasing microprocessor power consumption, Intel Technike Journal Q1 (2001).
[17] K. Skadron, M.R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, Temperature-aware microarchitecture, in: Proceedings of the 30th Intl. Symp. on Computer Architecture (ISCA), June, 2003.
[18] J. Srinivasan, S.V. Adve, P. Bose, J.A. Rivers, The Case for lifetime reliability-aware microprocessors, in: Proceedings of the 31st Intl. Symp. on Computer Architecture (ISCA), June 2004.
[19] X. Fan, C.S. Ellis, A.R. Lebeck, The Synergy between Power-Aware Memory Systems and Processor Voltage Scaling, in: Power Aware Computer Systems (PACS'03), Springer-Verlag, December 2003.
[20] Power Saving Techniques, Available from <http://www.book.opensourceproject.org. cn/embedded/oreillyembed/opensource/0596009836/id- i_0596009836_chp_14_sect_5.html.>
[21] E. Horowitz, S. Sahni, D. Mehta, Fundamentals of Data Structures in C++, W.H. Freeman, NY, 1995.
[22] W. Lu, S. Sahni, Low power TCAMs for very large forwarding tables, INFOCOM, 2008.
[23] Available from <http://www.bgp.potaroo.net/as1221/bgptable.txt>.
[24] Available from <ftp://pma.nlanr.net/traces/>.
[25] B. Talbot, T. Sherwood, B. Lin, IP Caching for Terabit Speed Routers, Globecom'99, pp. 1565–1569, December, 1999.
[26] D. Tarjan, S. Thoziyoor, N.P. Jouppi, CACTI 4.0 Technical Report, Available from <http://www.hpl.hp.com/techreports/2006/HPL-2006-86.pdf>.
[27] B. Agrawal, T. Sherwood, Modelling TCAM power for next generation network devices: in Proceedings of IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS-2006).