A Multizone Pipelined Cache for IP Routing

Soraya Kasnavi, Paul Berube, Vincent C. Gaudet, and José Nelson Amaral

Dept. of Electrical and Computer Engineering, University of Alberta
Edmonton, Alberta, T6G 2V4, Canada
kasnavi,vgaudet@ece.ualberta.ca
berube, amaral@cs.ualberta.ca

Abstract. Caching recently referenced IP addresses and their forwarding information is an effective strategy to increase routing lookup speed. This paper proposes a multizone non-blocking pipelined cache for IP routing lookup that achieves lower miss rates compared to previously reported IP caches. The two-stage pipeline design provides a half-prefix half-full address cache and reduces the cache power consumption. By adopting a very small non-blocking buffer, the cache reduces the effective miss penalty. This cache design takes advantage of storing prefixes but requires smaller table expansions (up to 50% less) compared with prefix caches. Simulation results on real traffic display lower cache miss rate and up to 30% reduction in power consumption.
Key words: IP lookup, IP Caching, Content Addressable Memory (CAM).

1 Introduction

The sustained increase in Internet traffic over the last decade has necessitated faster and faster backbone networks and a corresponding increase in network processor throughput. A fundamental task in routing IP traffic is finding each packet's destination address and corresponding next hop information in a routing table. Routing tables store routing prefixes, rather than full destination addresses, in order to reduce table size. Multiple prefixes may match a particular destination address. With Classless Inter-Domain Routing (CIDR), routing prefixes may have any length (0 to 32 for IPv4) [15]. In the case that multiple prefixes match an address, the correct lookup result is defined as the longest matching prefix. Consequently, routers must perform Longest Prefix Matching (LPM) when searching the routing table. Since LPM is performed in every router along the packet's path from source to destination, routers require a fast mechanism to perform the lookup in order to maintain high throughput and low latency under load.

An effective strategy to speed up routing lookup is to use a cache to store recent routing results for reuse. The performance of the cache depends on the characteristics of the IP traffic, such as its *temporal* and its *spatial* locality. Greater temporal locality increases the probability that destination addresses are frequently used, and thus increases the utility of a cached address. Spatial locality means referencing addresses in the same numerical range. When prefixes are cached, a single cache entry can cover a large number of destination addresses in a same numerical range. Therefore, the spatial locality in the traffic stream is converted to temporal locality in the cache access stream.

A cache naturally exploits temporal locality. However, routers manage traffic from a large number of hosts. In some cases, only part of the traffic has high locality. In a

router with a single cache low-locality traffic pollutes the cache with low-utility entries. These entries reduce the effectiveness of the cache for all traffic, and may cause thrashing. However, if the cache is split then the cache performance is improved [16, 5]. One portion of a split cache stores addresses or prefixes associated with shorter routing prefixes, and the other portion caches the addresses of prefixes associated with the longer routing prefixes. Such a *multizone* cache prevents the lack of locality in one portion of the traffic from polluting the locality in the rest of the traffic.

This paper proposes a novel multizone pipelined cache (MPC). We study a two zone MPC where one zone stores IP prefixes of 16 or fewer bits in length, while the second zone stores full addresses. This half-prefix half-full IP MPC has a lower miss rate when compared with full address caches. An MPC requires less lookup table expansion and on-chip area than full prefix caches. MPC uses a small buffer to store recent IP addresses that missed the cache. This non-blocking feature allows the cache to resume searching for other IP addresses during a miss. Hit-over-miss effectively reduces the miss penalty. Furthermore, vertical and horizontal pipelining reduces power consumption and increases throughput.

The remainder of this paper proceeds as follows: Section 2 discusses related work. Section 3 describes the cache architecture and cache operation. Consistency issues specific to IP prefix caching are discussed in Section 4. Results from simulation of the cache are presented in Section 5, and we conclude in Section 6.

2 Related Work

Many researchers have addressed the efficiency of routing caches. Some studied existing locality in IP traffic [9, 5]. Others designed more efficient caches for IP routing [6, 13, 16]. In 1988, Feldmeier demonstrated that a routing-table cache could reduce the lookup time in network gateways by 65% [9]. Berube *et al.* designed a method to implement a high density, fully associative CAM-based cache in the Xilinx VirtexE FPGA architecture [2]. Their design is further discussed in Section 3. Chiueh *et al.* designed a CPU style IP caching scheme and demonstrated that general-purpose processors can serve as a powerful platform for high performance IP routing [6]. Since the data streams presented to the network processors have very different characteristics than the streams accessed by general-purpose CPUs, the cache design must be considerably different and the cache coverage must be improved to achieve acceptable performance [4]. A network cache should cache address ranges rather than individual addresses and its blocks should be small. Liu proposes IP Prefix Caching to achieve lower miss rates due to higher locality in prefixes compared to individual addresses [13]. Prefix caching is very efficient due to increased spatial locality but the lookup table of prefix caches should be transformed to assure correct cache results [13, 16]. Section 4 presents this transformation. Cache miss rate can improve when the cache is divided into different zones dedicated to different lookup prefix lengths. Chvets and MacGregor studied a *multizone* cache [5]. They simulated a two-zone full address cache where IP addresses are stored in each zone according to their lookup result prefix lengths. Their new cache design shows miss ratios approximately one-half those of conventional caches.

IP caches have very large miss penalties because a miss requires a rather slow main table lookup. Complicated lookup techniques can be applied to the main table to increase the lookup speed. However, these techniques dramatically increase the table updating delays. Thus, improving the cache miss ratio could compensate for the large cache miss penalty and allow a simple main lookup table to provide fast table updates. Non-blocking general purpose processor caches hide memory latency by overlapping the processor computations with memory data accesses [3]. Special registers are used to hold information about each cache miss. The processor can then overlap the service of a cache read miss with the execution of the subsequent instructions [8, 11]. Bhuyan *et al*. used execution-driven simulation to study the impact of instruction level parallelism (ILP) and cache architectures on the performance of routers [12]. They observed up to 37% improvement for their traces due to multiple issues, out of order executions and non-blocking loads.

This paper proposes MPC, a multizone, non-blocking, pipelined cache. MPC uses prefix caching in multiple zone caches to improve cache miss ratio. MPC adopts a non-blocking buffer to reduce the effective cache miss penalty. A pipelined design implements a novel search and reduces power consumption. The details of the MPC architecture and features are further described in Section 3.

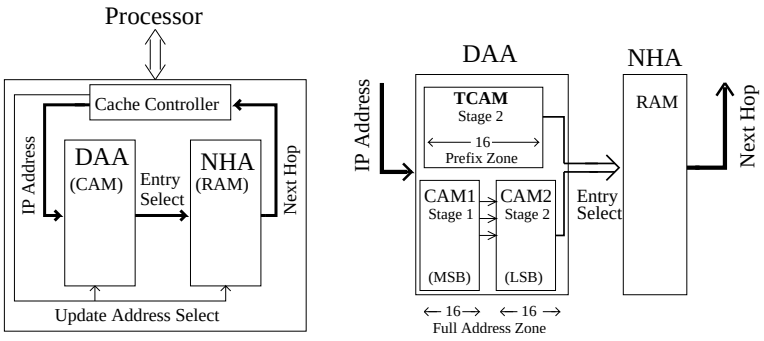
3 Architecture

Figure 1(a) presents a functional block diagram of a two zone MPC. MPC is based on the previous work of Berube *et al.* [2]. IP addresses or prefixes are stored in a *Destination Address Array* (DAA). Next hop information is stored in the *Next Hop Array* (NHA). The DAA and NHA are co-indexed, with one entry in the NHA corresponding to a single entry in the DAA.

MPC searches all entries of the DAA for an IP address. If the MPC finds the address in the DAA, a cache hit occurs, and the corresponding next hop from the NHA is returned to the processor. If no entry in the DAA matches the IP address, a cache miss occurs. In this case, a lookup in the full routing table is performed and the cache is updated with the new destination address/next hop pair.

Figure 1(b) presents a structural description of MPC. The DAA is divided into two parts horizontally. The two parts form the two zones of the cache, and have independent sizes. The upper *Prefix Zone* stores *short* IP prefixes, which are 16 or fewer bits long. The lower *Full Address Zone* stores full 32-bit IPv4 destination addresses. The *Full Address Zone* is further divided vertically, with each entry split in half. The most significant 16 bits of the address are stored in `CAM1`, while the least significant 16 bits are stored in `CAM2`.

The NHA is implemented using standard SRAM technology. The DAA is implemented using a combination of Content Addressable Memory (CAM) and Ternary CAM (TCAM) cells. A CAM is a fully associative binary memory capable of matching a specific pattern of data (a key) against all its entries in parallel. A CAM is used to implement the two halves of the *Full Address Zone*. A TCAM is used to implement the *Prefix Zone* of the cache because a TCAM can store *don't care* states in addition to 0s



(a) Functional View. (b) MPC Memory Allocation.

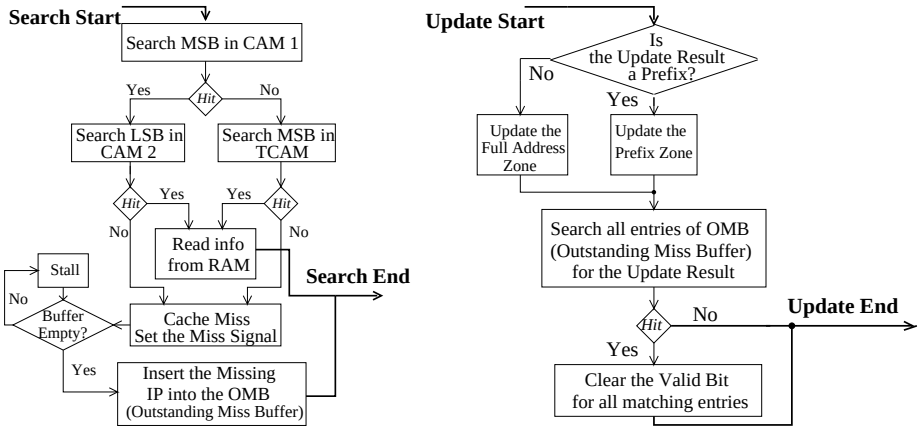
Fig. 1. General Description of the Cache Architecture.

and 1s. TCAM bits set to the *don't care* state will match both 0s and 1s in the key. Thus, a TCAM is well suited to store IP prefixes.

3.1 Cache Functionality

Breaking the DAA into three pieces allows cache lookups to be pipelined. The pipeline has three stages: (1) a lookup in `CAM1`; (2) a lookup in either `CAM2` or the *Prefix Zone*, as required by the results of the first stage; (3) an access to the *NHA RAM* (on a hit) to return the lookup result (forwarding information or cache miss indication). In stage 1 the most significant 16 bits of the address are applied to `CAM1`. If there are any matches in `CAM1`, in stage 2 the corresponding entries of `CAM2` are searched with the 16 least significant bits of the address to complete the full-IP match in the *Full Address Zone*. Otherwise, stage 2 applies the 16 most significant bits of the address to the prefixes cached in the *Prefix Zone*. If there is a match in either the *Full Address Zone* or the *Prefix Zone*, stage 3 accesses the RAM location corresponding to the matching entry, and returns the next hop data as the lookup result. Note that if the IP address hits CAM1, the prefix zone is not searched at all. The correctness of this lookup scheme requires that each IP address either hit the cache in the full address zone or in the prefix zone, but never in both. The routing table transformations required to ensure correct cache results are described in Section 4.

When there is no match either in the Full Address Zone or in the Prefix Zone, a cache miss is reported. A routing table search returns the routing information that is then stored in the MPC. The time required to complete this search and store the value in the cache is known as *miss penalty*. Servicing a miss is time consuming because of the slow main memory accesses to the routing table. MPC stores recent misses in an *Outstanding Miss Buffer* (OMB) until the processor returns the lookup results (see Section 3.2). Figure 2(a) depicts the pipeline flow diagram for a cache search. The diagram for an update is in Figure 2(b).



(a) Cache Search (b) Cache Update

Fig. 2. Flow Diagram of the Cache performance.

3.2 Outstanding Miss Buffer

MPC uses the OMB to store recent misses until the processor returns their lookup results. Without OMB, MPC would need to stall while each cache miss is serviced. Blocking hinders cache throughput because further cache searches cannot proceed until the lookup is performed and the cache updated, even if pending request would hit the cache. When a miss occurs in the non-blocking MPC, the address is stored in the OMB and the cache continues performing lookups. If subsequent IP addresses hit the cache while a miss is being serviced, a *hit under miss* occurs. A *miss under miss* (secondary miss) occurs when a subsequent IP address also misses the cache. Secondary misses are stored in the OMB until the buffer is full, at which point MPC blocks and the processor stalls until misses are serviced and removed from OMB. An example of MPC functionality with a two-entry OMB is given in Figure 3. In this example, IP2, IP4 and IP6 are cache misses. The MPC is able to search for IP3 and IP5 and forward their corresponding information to the processor while the main table lookup for IP2 is in progress. The MPC stalls after searching for IP6 because OMB is full. No new IP can be searched until IP2 is serviced and removed from OMB to make room for IP6.

When the lookup result of a pending IP address in the OMB comes back from the main memory lookup, the MPC updates either the prefix zone or the full address zone with the corresponding information according to the MPC replacement policy. An expansion of the lookup table ensures that the result is either a short prefix that updates the prefix zone, or a full address with 32 bits that updates the full address zone (see Section 4).

MPC requires a pipeline stall to update the data stage by stage. After an update is complete, the missing IP is removed from OMB. However, other pending IP addresses in the OMB might be identical to the recently updated IP address. Additionally, an

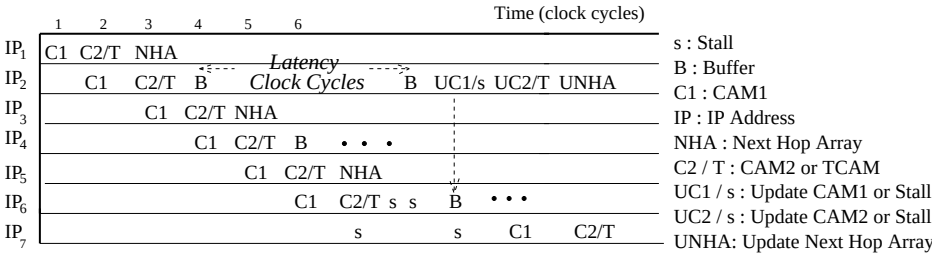
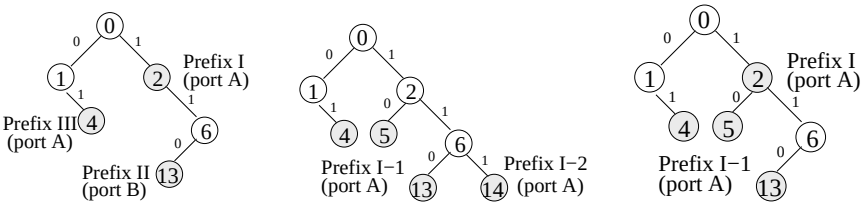


Fig. 3. Pipeline Diagram of the Cache.

update result might be a prefix covering multiple pending IP addresses in the OMB. To ensure that the same lookup result is not written into the cache multiple times, we implement the OMB as a 33-bit CAM. Each OMB entry stores a 32 bit address and a *valid bit*. Only valid entries require a software lookup and cache update. After each update, an associative search of OMB identifies all matching entries. If the lookup result is a prefix, its *don't care* bits are externally masked to ensure that they match with the data in OMB. The valid bits of all matching entries are cleared. A second search for those matching OMB entries will now hit the cache, and provide the processor with the next hop information. The flow diagram for an MPC update is shown in Figure 2(b).

4 Lookup Table Transformation

Caching prefixes reduces the cache miss rate because a stored prefix can cover a large range of the address space. However, routers are required to provide Longest Matching Prefix routing. If multiple prefixes match an address, a situation may arise where the longest matching prefix is not present in the cache, but a shorter matching prefix is in the cache. This short matching prefix will produce a cache hit, leading to an incorrect routing decision. Figure 4(a) depicts an example of three prefixes of a routing table organized as a *trie*. A trie presentation of a lookup table is a tree-based scheme where the root of the trie corresponds to the most significant bit of the address. Branching right indicates that a bit is 1, while branching left indicates that a bit is 0. A complete trie enumerates every possible address. In order to reduce the space requirement of the trie, only the nodes required to form a path to each prefix are stored. Nodes are sequentially numbered from top to bottom and from left to right. Gray nodes represent prefixes stored in the lookup table. The prefix in node 2 (prefix I) is said to encompasses the prefix in node 13 (prefix II), because node 2 is on the path from the root to node 13. Encompassing prefixes are responsible for situations where a prefix cache may produce incorrect routing decisions. If Prefix I is cached, it will match with IP addresses whose longest matching prefix in the trie is Prefix II. An IP address matching Prefix II could then be incorrectly matched by Prefix I and forwarded to port A. Therefore, encompassing prefixes are *non-cacheable*. In this example, the prefix at node 4 (prefix III) is *cacheable* because it does not encompass any other prefix.



(a) Original trie (no expansion)

(b) Full trie expansion

(c) Partial trie expansion in [13]

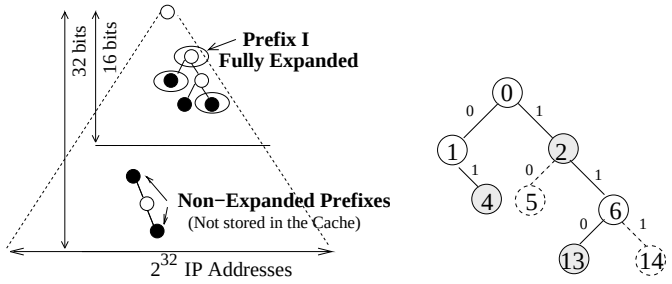
Fig. 4. Trie presentation of a small lookup table.

4.1 Prefix Caching and Table Expansion

One general solution to ensure correct routing cache results is to cache only *cacheable* prefixes and cache full IP addresses when lookup results are *non-cacheable* prefixes [13, 16]. This solution requires no lookup table transformation, but the cache miss rate will be higher because of the reduced coverage afforded by the full addresses placed in the cache. Moreover, the lookup scheme must decide if a prefix is cacheable or not. Other solutions expand the trie to ensure that all prefixes are leaves of the trie, and thus *cacheable*. Figure 4(b) shows a complete expansion of the trie for our example. Prefix I is expanded by appending ”0” to form prefix I-1 in node 5, and by appending ”11” to form prefix I-2 in node 14 and the forwarding information (port A) is copied in both nodes 5 and 14. Observe that the number of valid prefixes increases and the lookup table gets larger. Liu reports up to an 118% increase in table size [13]. Routing table expansion is unfavorable due to memory area limitations, power consumption and cost. On the other hand, table expansion pushes prefixes lower in the trie, increasing the search time for Software searches. Also, updates in a fully expanded table are challenging, since an expanded prefix no longer exists and the update must find the prefixes created by expansion. Liu presents a partial prefix expansion to reduce the number of prefixes in the lookup table for a prefix cache [13]. Figure 4(c) depicts the partial expansion in [13] where non-cacheable prefixes are expanded only to their first level of expansion. Prefix I-1 is added to the trie at node 5 with same forwarding information as prefix I. Observe that routing table growth is less than with full expansion, but the lookup scheme must still decide if a prefix is *cacheable* or not.

4.2 Table Expansion in MPC

Our MPC solution for table expansion is illustrated in Figure 5(a). MPC fully expands the routing table for short prefixes, pushing them down the tree until they either become leaf nodes, or become 17 bits long. All short prefixes are thus cacheable in the Prefix Zone of the MPC. Any destination address matching a long prefix in the lookup table is stored in full in the Full Address Zone of the MPC. This table transformation has the following advantages:



(a) The Proposed Partial Table Expansion)

(b) Expansion Free Transformation

Fig. 5. Table Transformation.

1. Routing table expansion is limited to those prefixes that provide the greatest coverage of the IP address space.
2. At most one prefix stored in the Prefix Zone can match a destination address, since all short prefixes are cacheable.
3. Since the short prefixes in the Prefix Zone are cacheable, no address can hit both the Prefix Zone and the Full Address Zone. Therefore, any address that could hit the Prefix Zone is guaranteed to miss the Full Address Zone.
4. A length check is sufficient to determine if a prefix is cacheable.

Note that points 2 and 3 guarantee that there cannot be multiple hits for a search in the cache, which simplifies cache design. Also, point 3 enables our power-saving pipelined search.

4.3 Expansion-Free Software Lookups

Lookups may be implemented in hardware or software. A software lookup walks down the trie to find the longest matching prefix for an IP address [15]. The longest matching prefix is the last prefix encountered in the trie. Some of these longest matching prefixes might not be cacheable in a non-expanded table. We propose a new *Expansion-Free* (EF) method to generate cacheable prefixes using a simple and inexpensive mechanism during a software lookup. EF forwards the generated cacheable prefixes to the cache but does not store them in the lookup table, thus eliminating problems associated with table expansion. Since the original submission of this paper, we learnt that Akhbarizadeh *et al.* independently developed a similar method [1]. Figure 5(b) illustrates the EF method on the example of Figure 4. Let n be the last node visited during a traversal of the trie for an IP address, and let p be the last node visited containing a prefix during the traversal. EF has three rules.

1. If $p = n$ is a leaf node in the trie, then the prefix in p is cacheable and can be forwarded to the prefix cache. For example, for IP addresses covered by node 4,

$p = n = 4$. Thus the prefix in node 4 is cacheable and is forwarded to the cache as the lookup result.

2. If $p = n$ is not a leaf node in the trie, p is not cacheable. In Figure 5(b), assume an IP address matches node 5, $n = p = 2$. A cacheable prefix can be produced by adding 0 (the next bit in the address) to the path followed to encounter p. This generated cacheable prefix is forwarded to the cache.
3. If $p \neq n$, the prefix in p is not cacheable. In Figure 5(b), assume an IP address matches node 14 in the trie. For this IP address, $p = 2$ and $n = 6$. A cacheable prefix is produced by adding a 1 (the next bit in the address) to the path traversed to find n. This generated cacheable prefix is forwarded to the cache.

5 Performance Evaluation

To evaluate the MPC design we build a high level architectural simulator and run it with real IP traces and lookup tables of three distributing (neither core nor edge) routers of local service providers. The characteristics of the data used for simulations are given in Table 1.

Table 1. Trace Characteristics

	ISP1	**ISP2**	**ISP3**
Trace Length (Packets)	99117	98948	98142
Routing Table Size (Prefixes)	10219	10219	6355

To evaluate the performance improvements achieved by MPC, we compare the miss rates of MPC with a full address IP cache and a full prefix cache. For a fair comparison, the IP cache is simulated as a two-zone two-stage pipelined cache with equal sized zones. This architecture caches full IP addresses and is implemented in a 32 bit binary CAM. The prefix cache is a 32-bit Ternary-CAM that store prefixes, using a fully expanded version of the real lookup table (LUT). The miss rates are given in Table 2.

Table 2. Miss Rates (%) vs. Cache Sizes (No. of Entries) for Three Traces

	ISP1			**ISP2**			**ISP3**		
Entries	**IP**	**Proposed**	**Prefix**	**IP**	**Proposed**	**Prefix**	**IP**	**Proposed**	**Prefix**
512	22.7	15.5	7.4	10.8	6.2	2.9	3.6	3.0	0.5
1024	15.4	7.9	2.5	7.2	3.3	1.3	2.2	2.0	0.5
2048	10.5	3.7	1.4	4.9	2.0	1.2	1.9	1.6	0.5

Clearly, the prefix cache outperforms the two other caches with the same number of entries. However, the prefix cache must be implemented in a 32-bit TCAM. Since

the area required for a TCAM cell is almost twice the area of a CAM cell, MPC and the IP Cache use half the area of a prefix cache with the same number of entries. To compare the performance of caches with the same storage area, MPC and the IP cache should be compared to a prefix cache with half as many entries. The simulation results indicate that for equal cache size (storage area), the performance of MPC is almost as good as the prefix cache. Moreover, the prefix cache requires full LUT expansion while MPC requires a partial expansion (as described in Section 4). Table 3 compares the total number of prefixes in the LUT after the expansion for a prefix cache and MPC.

Table 3. Number of Prefixes after Table Expansion

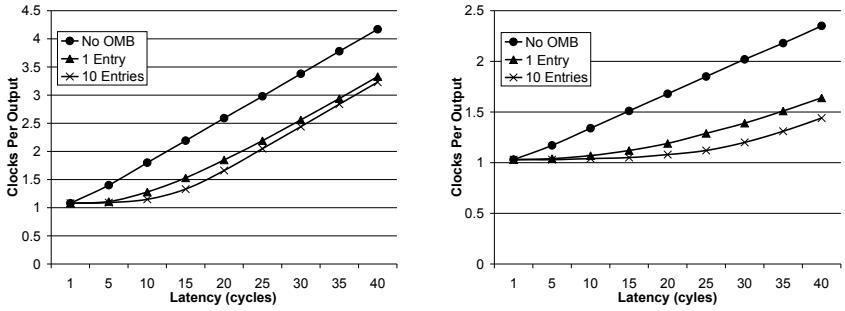
	ISP1		**ISP2**		**ISP3**	
	Entries	% Larger	Entries	% Larger	Entries	% Larger
Original Table	10219	–	10219	–	6355	–
Prefix Cache	30620	199	30620	199	7313	15
MPC	17485	71	17485	71	6469	2

MPC uses a small buffer (OMB) to hide the miss penalty. The miss penalty is modeled in our simulator by a *latency* parameter. A cache that has no buffer to store recent misses has to stall at each miss and wait until the update result is returned to the cache. To evaluate the impact of the miss penalty we measure a metric called CPO (Clock Per Output) that reports the average number of clock cycles necessary to provide the Next Hop Information for an IP address. Figure 6 depicts CPO versus Latency for MPC with no OMB, OMB with a single entry, and OMB with 10 entries. As expected, CPO increases linearly with latency for a cache with no buffer. For small latencies, in an MPC with a single entry OMB, the CPO is almost independent of the latency. For larger values of latency, CPO again increases linearly, but remains less than without the OMB.

5.1 Power Savings

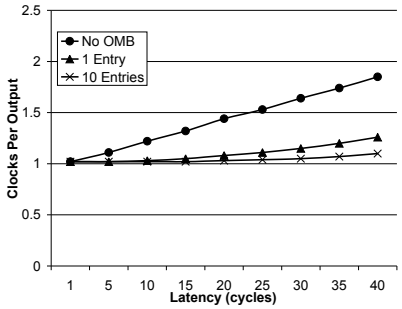
In a CAM-based device, power consumption is an important constraint that is addressed by many designs [7, 14]. The power consumption in a CAM-based device can be separated into three components: Evaluation Power (Search power), Input Power and Clocking Power [10]. All these sources are linearly dependent on the number of entries searched. If 50% of the time, only half of the entries of the cache are searched, the effective number of entries during each search operation is reduced to 75% of the physical number of entries. Thus 25% power is saved.

MPC divides the cache entries into two zones: the TCAM prefix zone and the CAM full address zone. For the study of power consumption, we assume that each zone contains half of the cache entries. The prefix zone is searched only if the address misses CAM1 of the full address zone. Our simulation results, presented in Table 4, indicate that almost 60% of the IP addresses hit CAM1, eliminating the need to search the TCAM. This results in a 30% reduction in the effective number of entries searched



(a) ISP1

(b) ISP2



(c) ISP3

Fig. 6. CPO vs. Latency for 1K-Entry (equally sized zones) MPC.

in the cache, and a corresponding 30% power savings compared to caches that search all entries.

6 Conclusion

We have proposed MPC, a non-blocking, multizone-pipelined-cache that dedicates different zones to different lookup prefix lengths. The *Prefix Zone* is able to store and search prefixes with 16-bits or less. The *Full Address Zone* stores and searches for full IP addresses whose lookup prefixes are more than 16 bits long. Prefix caching increases the cache coverage while a relatively small table expansion is required. EF method, proposed in this paper, completely eliminates table expansion for software lookups. Also MPC potentially can achieve higher throughput and low power consumption due to pipelining. The effective miss penalty is also reduced by using the non-blocking buffer to let the cache search for new IPs while waiting for the lookup results of cache misses.

Table 4. CAM1 Hit Rates

# Entries	ISP1 %	ISP2 %	ISP3 %
512	62	64	74
1024	63	65	74
2048	63	65	74

References

1. M. J. Akhbarizadeh and M. Nourani. Efficient prefix cache for network processors. In *12th Annual IEEE Symposium on High Performance Interconnects*, pages 41–46, Aug 2004.
2. P. Berube, A. Zinyk, J.N. Amaral, and M. MacGregor. The bank nth chance replacement policy for FPGA-based CAMs. In *13th International Conference on Field Programmable Logic and Applications (FPL)*, Lisbon, Portugal, September 2003.
3. T. Chen and J. Baer. Reducing memory latency via non-blocking and prefetching caches. In *5th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pages 51–61, Oct 1992.
4. Tzi-Cker Chiueh and Prashant Pradhan. Cache memory design for network processors. In *Sixth International Symposium on High-Performance Computer Architecture*, pages 409–419, Toulouse, France, January 2000.
5. I.L. Chvets and M. MacGregor. Multi-zone caches for accelerating IP routing table lookups. In *Merging Optical and IP Technologies Workshop on High Performance Switching and Routing*, pages 121–126, May 2002.
6. Tzi cker Chiueh and Prashant Pradhan. High performance IP routing table lookup using CPU caching. In *IEEE INFOCOM (3)*, pages 1421–1428, 1999.
7. A. Efthymiou and J.D. Garside. A CAM with mixed serial-parallel comparison for use in low energy caches. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, 12:325–329, March 2004.
8. K. I. Farkas and N. P. Jouppi. Complexity/performance tradeoffs with non-blocking loads. In *21st Int. Symposium on Computer Architecture*, pages 211–222, 1994.
9. D.C. Feldmeier. Improving gateway performance with a routing-table cache. In *IEEE INFOCOM 88*, pages 298–307, March 1988.
10. C. Jen; H. Hsiao, D. Wang. Power modeling and low-power design of content addressable memories. In *IEEE Int. Symposium on Circuits and Systems*, pages 926–929, May. 2001.
11. D. Kroft. Lookup free instruction fetch/prefetch cache organization. In *8th Int. Symposium on Computer Architecture*, pages 81–87, May 1981.
12. H. Wang L. Bhuyan. Execution-driven simulation of IP router architectures. In *IEEE Int. Symposium on Network Computing and Applications*, pages 145–155, Oct. 2001.
13. H. Liu. Routing prefix caching in network processor design. In *Tenth International Conference on Computer Communications and Networks*, Oct 2001.
14. K. Pagiamtzis and A. Sheikholeslami. Pipelined match-lines and hierarchical search-lines for low-power content addressable memories. In *IEEE Custom Integrated Circuit Conference*, September 2003.
15. M.A. Ruiz-Sanchez, E.W. Biersack, and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. *IEEE Network*, 15:8–23, April 2001.
16. W.L. Shyu, C.S. Wu, and T.C. Hou. Multilevel aligned IP prefix caching based on singleton information. In *GLOBECOM 02*, volume 3, page 2345 2349, Nov 2002.