A Pipelined IP Forwarding Engine with Fast Update

Yeim-Kuan Chang, Yen-Cheng Liu, and Fang-Chen Kuo
National Cheng Kung University, Tainan, Taiwan
{ykchang, p7695452, p7895107}@mail.ncku.edu.tw

Abstract

IP address lookup is one of the most important functionalities in the router design. To meet the requirements in high speed routers consisting of line-cards with 40Gbps transfer rates, researchers usually take lookup/update speed, storage requirement, and scalability into consideration when designing a high performance forwarding engine. As a result, hardware-based solutions are often used to develop a high speed router nowadays. In this paper, we develop a FPGA-based pipelined forwarding engine which focuses on reducing the update overhead. The proposed scheme partitions the routing table into several disjoint groups. The prefix which resides in the same group is interleaving stored into several memory modules to ensure the parallel comparison at the comparison stage. With the pipeline enabled, the throughput of the design can achieve the speed of OC-768. The update overhead can also be reduced.

Keywords: IP lookup, pipeline, route update, FPGA

1. Introduction

An internet router is a computer network device, called Layer-3 switch, which can be seen as a switching node of multiple networks or logic subnets. Internet traffic is doubled every three month, and the internet hosting is tripled every two years according to results presented in [6] and [9]. Furthermore, more and more network applications have been proposed in the new era, such as peer to peer file sharing and streaming, real time video, network games, etc. These new applications require more bandwidth. Therefore, a good design of the high performance internet router is demanded greatly.

There are several tasks in the Internet routers. One of the critical goal of the Internet routers is to forward the Internet packets to its destination port by processing the IP destination address which is stored in the packet header. As the growth of internet service is increased rapidly nowadays, the forwarding engine should be ready to handle the gigabit-per-second traffic rate. Several methods including hardware-based manners are proposed by researchers in the recent years to achieve the high performance forwarding engine.

Hardware-based methods which are based Application Specific Integrated Circuit (ASIC), Field Programmable Gate Array (FPGA), Ternary Content Addressable Memory (TCAM), and network processor schemes often handle packet forwarding and update problems in the high speed forwarding engines. Network processor, ASIC and FPGA based methods usually store the routing table into static RAM (SARM). TCAM can store not only 0 and 1 but also "don't care" value on the cell. Researchers can select one of them suited to their design.

As we already know, if we try to store the entire data structure in a single port SRAM memory block by software codes, processing a lookup or an update operation (include insert and delete) may cause multiple memory accesses (one access per one level) to complete the task. The phenomenon is one of the primary reasons that slowdown the lookup speed. Therefore, researchers start to design an embedded pipeline technique into forwarding engines to increase the throughput. By applying pipeline system, one packet can be forwarded per one clock cycle.

Several pipeline forwarding schemes were proposed in the past. Most of the designs are based on ASIC or FPGA. Some of them estimate the results by some famous tools, for example, CACTI [5]. Therefore, in this paper, we will not only focus on developing a suitable data structure for the FPGA-based IP address lookups with incremental update but also implement it in the real hardware. Meanwhile, we will take a look on other existing methods, and try to make a fair comparison with other schemes in the paper.

The rest of the paper is organized as follows. In section 2, we focus on other scheme which developed by other researching groups. The section 3 illustrates the proposed design. In section 4, we present our experiment results. The last section is the conclusion.

2. Related works

There are several approaches to matching an IP-address against a group of prefixes constructed by routing tables. Searching bit by bit is called trie-based lookup. A trie structure is a tree–based data structure which allows the organization of prefixes on a digital. It's a common structure for IP lookup scheme.

Binary trie is a basic trie structure for lookup. Generally, the search complexity of binary trie is O(L), where L is the length of address. Multi-bit tries are often applied to develop forwarding engine. The major benefit is that memory access time can be reduced due to tree depth reduced. Level compressed (LC) trie, which proposed in [8], can be seen as an adapted scheme to the Multi-bit tries and binary trie. LC trie can reduce more memory and the number of memory access.

Trie-based structure is well-known suitable for applying pipeline IP address lookup. The main concern of these kinds of architecture is to figure out a technique to balance the memory distribution across stage. The reason is that in each pipeline stage, the critical path of the design is composed by two periods.

The first period is for logic processing, which include the shifting for the offset in the array if multiple-bit trie style are applied in the design, adding the offset with n-bits register which store the index address of the for finding out the real address in the physic memory, and make a decision if the lookup is already in the leaf node. Another period is for the memory accessing, which obtain the information of next address in the SRAM that which imply the initial index of the next stage ram. As we know, the logic processing time in each stage is nearly the same. Therefore, the memory accessing time becomes the critical issue in the pipeline forwarding system. Figure 1 illustrates a simple example of the design.

Multi-bit trie is the common applied data structure for pipeline forwarding engine. It can not only improve the throughput but also can reduce the search delay compared with binary trie. In the following we will introduce two schemes which using pipeline to improve the throughput.

2.1. Ring Pipeline

In [1], Baboescu *et al.* proposed a new architecture called random ring pipeline to minimized total memory cost and maximized packet forwarding throughput. The design aims on balance the memory usage in every stage as it is the critical issue we mentioned above. The design allows multiple entry points to achieve the goal

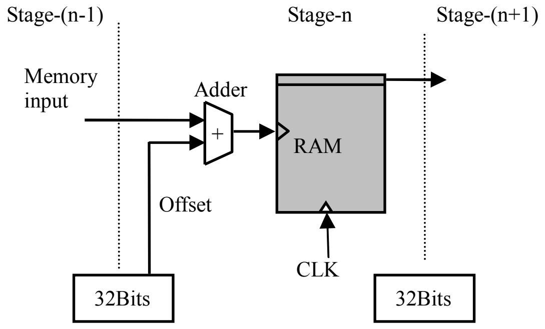

Figure 1. illustrates a simple example of the design.

of the memory balance. In other words, every stage can be used as the starting stages of any packets for the IP address lookup.

The approach is starting by partition the original binary-trie data structure into several sub-tries. The number of sub-tries indicates the number of entry point of the design. The second step of the method is to partition each sub-tires into several levels. The last step is to map these levels into the stages in the pipeline system based on the memory balance constrain across the stage.

With the fashion of the design, it will encounter the problem of contention. To solve the issue, the authors use two data paths. The first data path is active during the odd clock cycles and it is used for a first traversal of the pipeline. The second data path is traversed during even cycle, it will allow the task which is left after first path processing to continue its execution on the pipeline stages. The authors claim that three issues are guaranteed. The first one is that the output rate is equal to output rate. The second is that all of the packets left the engine in order. The last one is that all of the lookup tasks have the constant delay. Therefore, the throughput of the design is the frequency/2. The delay of a lookup operation for a packet is doubled. Figure 2 is the example of the ring pipeline design.

2.2. Circular Pipeline

In paper [7], the author Kumar proposed another pipelined packet forwarding architecture called Circular, Adaptive and Monotonic Pipeline (CAMP). The scheme aims on develop a more flexible method to mapping the trie node in either binary trie or multi-bit trie into particular stages by free the two limitations in order to obtain a more balanced memory distribution across the stages. The first one is that the pipeline is not necessary. Unlike the above two schemes, CAMP provides multiple entry points and multiple exiting points.

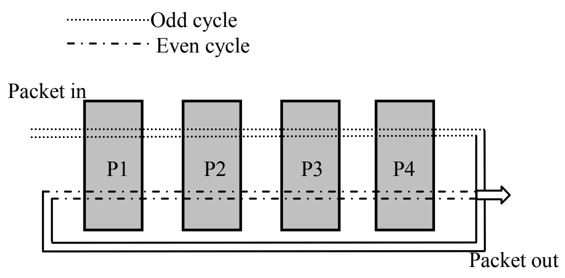

Figure 2. Ring pipeline design.

The first step of the approach is the same with ring pipeline. It uses several initial bits to get the hash value to decide which particular stages as its entering stage. In the next step, the authors apply coloring and min-max heuristic to assign the remaining sub-trie into particular stages. It aims on obtain a balanced color usage by coloring the sub-tries in a sequence such that the larger sub tries are colored before the smaller ones. After mapping the node is complete, a reordering process is enabled if necessary.

There are three main properties of the design. Firstly, It allows dynamic entry and exit points. Secondly, it is circular, thus all neighboring stages are connected in the same direction. Thirdly, it supports no-operations for which requests are simply passed over whenever the designated node is not found. The throughput of the design is 0.8 × frequency in the worst case because that the contention problem will occur. The request queues before the entering the system are applied to reduce the problem. However, the contention situation will happened when the lookup per clock exceed 0.8. Figure 3 is the brief scheme of the design.

3. Proposed Scheme

Although the traditional multiple-bit trie is suitable for the packet forwarding, there are several problems while fast update operations are considered.

First, multiple memory elements in each updated stage might be performed while update bubbles arrive at the stage due to the prefix expansion required for constructing the multi-bit trie. In other words, multiple memory writes are needed to complete the task. Second, according to the distribution of the well-known routing tables, the prefixes of some particular lengths, such as "24", dominate in the routing tables. This phenomenon will cause the memory usage imbalance among pipeline stages. Based on the update design in [2], prefixes which locate in the same level should be mapped to the same stage. It will be difficult to combine the update messages in different stages because the dominance of some prefix lengths described. Therefore, researchers should not only try to

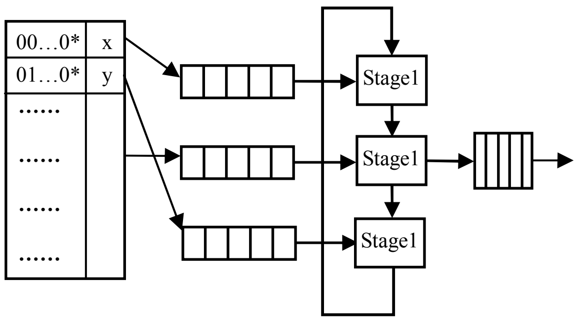

Figure 3. The example of the Circular, Adaptive and Monotonic Pipeline

balance the memory across the stages, but also balance the update frequency across the pipeline stages.

Third, in [2], the authors decide to map the nodes to particular memory stages by their levels in the binary trie. With the constraint, the implementation will be simple. However, to balance the memory across the stages will be difficult because the prefix distribution is always imbalanced. Last, since the update bubbles constructed by software will interrupt the lookup process, the throughput of the forwarding engine is going to decrease, especially in the environment of high update request rate. Ring pipeline and circular pipeline can make a more balanced distribution of the memory across the stages. However, the two schemes still have problem on updating process due to their design style. Therefore, we propose a scheme called *Enclosure Grouping* (EG) to reduce the problems mentioned above.

3.1. Observation of Prefix

The Border Gateway Protocol (BGP) can be seen as the core of the internet. The main goal is to perform inter-domain routing in TCP/IP network. Based on the idea presented in [4], with the observation of the prefixes we can briefly classify these prefixes into several groups in the routing tables by prefix enclosure property. For instance, prefix "A", 140.116.82.30/26, is covered by prefix "B", 140.116.0.0/16. Based on the prefix enclosure property, the group which contains the prefixes that do not cover any prefix (the most specific ones) is named Level 1. The group which contains the prefixes that only cover prefixes in level 1 is called level 2. In general, the group which contains the prefixes that only cover prefixes in level 1 to $k - 1$ is called level k. Notice that the prefixes in the same level will be disjoint with each other. As the experiment result showed, more than 90% of prefixes reside in level 1.

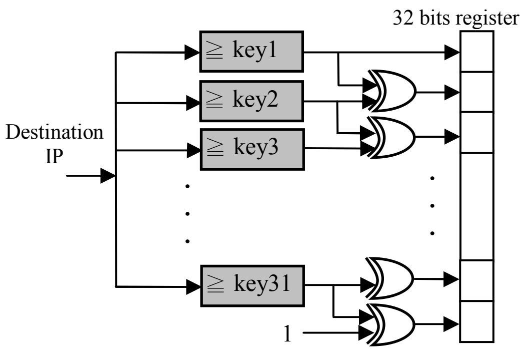

Figure 4. The N-way range comparator.

3.2. Range Partition

With the level partitioning scheme described above, we further partition the prefixes in each level into small subgroups to increase the search speed. We use the range partition that employs the 32-bit IPv4 or 128-bit IPv6 addresses as our partition key to evenly separate the prefixes in each level into several balanced subgroups. With the experiments shown in the real world routing tables, range partition is a good approach to obtain a uniform distribution of the prefixes. Since the prefixes in the same level are disjoint with each other, we will simply apply the in-order binary trie traversal to accomplish the partition task.

As the bucket size is almost full, the partition key can be selected. Care should be taken for choosing the keys because a poor selection might degrade the update operations. The following two issues are the major concerns of the key selection. (1) To reduce the prefix duplication, we have to reduce the chances that the chosen key covered by any prefix. The other shortcoming is that it might result in two memory writes. (2) If (1) us happened, the chosen keys should be able to divide the prefix which covers the key into two prefixes.

Based on the two concerns, we choose the endpoint of the prefix as the partition key in the design. One selection is to apply the high end point. A 32 bits comparator and a "larger" operator are required for one range in the scenario. Another one is to apply low endpoint of the prefix. In this case, an n-bits comparator, where n ranges from 8 to 32, and a "larger-equal" operator are need. Thanks to the length distribution of the routing table, n is 24 for most of the comparator. Theoretically, a bigger n-bit comparator will result in more resource used. Therefore, we choose low endpoint as the partition key in our design. By using the endpoint as the key, issue (2) is no longer a problem. On the other hand, chances that (1) takes place will decrease because the prefixes in the same level are disjoint. Consequently, (1) will not happen before the prefix is deleted or pushed down to the lower levels.

3.3. Find the search group

The number of groups, N, in each level will depend on the total number of the prefixes in the level. Therefore, we should be careful with the tradeoff between group size and the number of the groups. The reason is that the resource used for deciding which group is going to be searched for the incoming destination IP address is one of the main design costs. It should be noticed that we always define N as the power of 2 to reduce the complexity of the design. Figure 4 is the simple model of our N-way range comparator.

As the figure shows, when the input packet enters the engine, the range comparators will compute the destination IP address to figure out the group that the incoming packet might have the opportunity to match. In our design, we will use two levels to partition. The range key in the first stage is designed to be hardwired values. Therefore, they can not be changed except when the system is reconfigured. We will choose 31 keys in the first stage of our design. The reason we do not select more keys is that we should consider the fan-out problem which larger fan-out might cause longer route delay. After processing the first stage, there is only one bit from the 31 bits registers will be set to enable in each level.

After the first stage partition, we like to partition the prefixes into smaller subgroups. The second stage of our partition key is stored in the on-board RAM instead of hardwired values in the first stage. One advantage of this two-stage partitioning scheme is that the resource usage can be reduced. Another advantage is that the range key can be changed so that the flexibility of design increases. For instance, if we want to partition the prefixes into N groups, $N - 1$ comparators are required. By storing the range key in RAM, only $2\times\sqrt{N}-1$ comparators with some encoders are needed in our architecture. The encoders are used for determining the memory address where the second stage range keys reside. The range keys are put interleaving in $\sqrt{N}-1$ memory blocks. Therefore, in order to make the processing time nearly equal to stage1, we will divide the stage into 2 stages to reduce the critical path in the step.

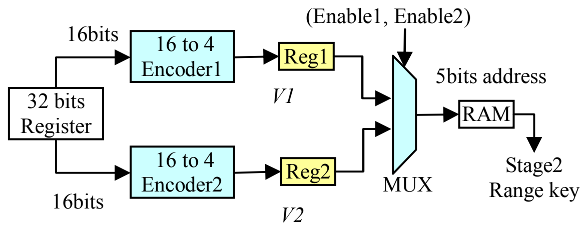

Figure 5. Encode address to access RAM.

In the first stage of encoding step, two 16 bits to 4 bits encoder with one enable bit is applied. Each of the encoder serves 16 bits input, and it will generate a value *V*, which is range from 0 to 15. Meanwhile, the enable bit will be set to 1 if one of the 16 bits is nonzero. The second stage of the encoding step requires one 2 to 1 selector to generate the real address, *Addr*. These selectors will use the enable bits which are the output from last stage as the index and *V* to obtain the output value of the stage. Figure 5 illustrates a simple example of the design.

With the memory address we mentioned, $\sqrt{N}-1$ range keys are output to compare with the destination IP address to find out the subgroup which the prefix in the subgroup may have the chance to match. After the comparison, another $\sqrt{N}$ bits registers are stored. These bits are again being encoded to a memory address with the same process we introduce above to get the memory address where the prefixes reside.

3.4. Memory Design and Match Process

With partitioning the prefix in the level into *N* uniform groups, the size of the prefix in each group will become smaller. Therefore, instead of TCAM-based method which usually stores the prefix in the same group into the same block, we will interleave store the single group into the same memory address in several different single port SRAM memory modules. In other words, the depth of each memory module in our design is *N*. In order to achieve the goal that whole prefixes in the same group can be read out in only one memory access for paralleling comparison, there is only 1 memory address which stores the prefix in the same group in the depth *N* memory module.

Figure 6 briefly presents an example of the single memory block of our design. In each of the address line, 37 bits is allocated for 1 prefix. The 37 bits element can be separate into 2 parts. The first part of the memory element is allocated 32 bits, which only store the value, *P*, that high endpoint of the prefix with shift the value, M. The second part is 5 bits, which store the value *M* (32 – length of prefix). We store the

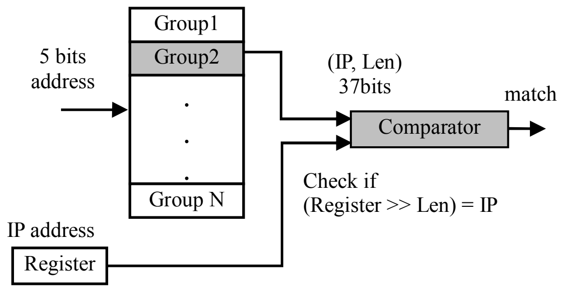

Figure 6. Example of the single memory block

value *M* instead of length of prefix because it can reduce the processing time on comparing the prefix with the incoming destination IP address. After the memory access, the data will be compared to the IP address.

3.5. Overall Architecture

After the processes above, the search operation is done in each which group of prefixes that the destination IP address may match. The method we apply is the range partition. In the stage2 and stage3, the goal is to encode the enable bits for the memory address where the range keys are stored. We separate the encoding step into several distinct stages because the number of bits we try to encode is very large. Stage4 to Stage8 is the second step of range partition and find out the memory address where the prefixes are stored. In the Stage9, the comparison between prefix and input destination IP is performed. Through these stages, an incoming packet can determine if there is a prefix match in the routing table.

As we already know, there are at most 6 levels in the IPv4 routing tables. Due to the reason that the goal of IP lookup problem is aim on finding the longest prefix match, a priority encoder is needed to complete the search operation which that level 1 is the highest priority.

As we have mentioned above, the number of levels are at most 6 in most of IPv4 tables. The straightforward way of our architecture is to design 6 engines which we introduced above. The drawback of the design is that the hardware cost is high because each level requires several range comparators. Therefore, we provide a technique to reduce the hardware overhead.

Since the sizes of levels 3 to 6 prefix are small, we can combine all levels except level 1 into one group. With leaf pushing, all the prefixes in the new group are still disjoint with each other. We also separate the prefix with length 8 as another group to reduce the expansion prefix by leaf pushing. These prefix are

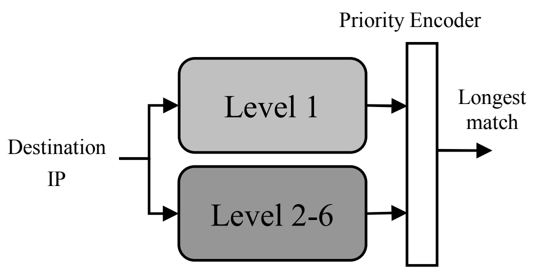

Figure 7. Overall architecture.

store in the 8-stride expansion array. The shortcoming of the design is that memory usage will increase a little, and it will take more cycles to complete a route update in the combined level. However, it can reduce the hardware cost of our design. Therefore, a tradeoff between the hardware cost and update performance can be considered. Figure 7 is the brief view of the architecture.

3.6. Update Scenario

The route update occurs when the update massages come to the routers. In our scheme, most of the update operation can be done in only one clock cycle. The reason is that all of the prefixes in the same level are disjoint with each other. Although the prefix in different levels might have enclosure relation, the route update still can be done in a single cycle because different level of prefix will be stored in different memory banks. The situation that update process require more than one clock cycles to be completed in level 1 is that the “add” operation and the prefix to be added covers the partition key, which we have describe above. However, the probability of the situation is very low. It can also be prevented by storing the expanded prefixes into two memory block because these two new prefixes is belong to different groups in the situation. Therefore, our proposed scheme takes advantage of the update operation than multiple-bit trie pipeline schemes. The following we will introduce more detail of the update process of the scheme.

In the update process, the aids from software system are required. It should be noticed that in a pipeline IP forwarding scheme, the pre-compute for update operation is required. Therefore, the effect of update operation to the search engine is based on number of time that search engine being interrupted. For instance, in [3], the authors claim that reduce the bubbles and keep the search engine active are relatively important than completing the update quickly. The software system should record the 3-tuple for the update demand. The first value in the 3-tuple is to compute which group the updated prefix belongs to. The second value is to compute which memory block is free for allocating the new prefix. The third value is to compute which memory block the prefix belongs to. Now we separate the update operation into add and remove respectively.

Insert: The first two values are applied when the new prefix is to be added to the routing table. If the newly added prefix does not have enclosure relationship with the present prefix, the newly prefix is simply inserted to the free location of the memory. If there is enclosure happened, the software aid should compute another 3-tuple for the next level to complete the update.

Delete: The third value is employed when one prefix is going to be deleted. It also needs the first value. It should be noticed that the removed prefix may reside in two groups as we mention above. Therefore, when the situation happened, another 3-tuple is required to complete the operation.

There are also some problems may occur on updating. For instance, the size of particular group becomes much larger than other groups. One way to solve the problem is to pre-allocate more memory space for each group. The other way is to reconfigure the partition keys or hash function. In our propose scheme, we will provide another approach to solve the problem. Before we introduce the technique, we will define a term as follows.

Definition: A prefix which does not cover or be covered by other prefixes is called *Stand alone prefix*.

Based on the level definition we describe in section A, the stand alone prefix is initially assigned into level 1. As we know, most prefixes are resided in level 1, which means most route update occurs in level 1. Therefore, to reduce the overhead of memory allocation in particular group, we can move the stand alone prefix, especially with short length of prefix, in the group to lower levels. The reason is that stand alone prefix is disjoint with other prefix. With the technique, we can temporary solve the memory allocation overhead. Our proposed method also has another advantage on update. With the prefix in the same group are stored in different memory module interleaving, the update process will not interrupt the search operation.

4. Experiment Results

Our simulation results are programmed by two programming languages. With the implemented in C language, we use GCC-3.2.2 as the compiler. The result is proceeding on the Intel Pentium-IV 2.4GHz with 8K L1 cache and 512KB L2 cache. The operation system is Debian GNU/Linux 3.1 (Sarge) (Intel X86 (32-Bit OS)). The hardware simulation of our design is

Table 1: The comparison of our scheme and the other two scheme.

Method	Throughput (Gbps)	Memory (MB)	Slice
Proposed EG	70	1.25	4094
Ring Pipeline 24 stages	40	3.49	408
Circular Pipeline 24 stages	64	3.88	528

Table 2. The conjecture based on the simulation of original multi-bit trie and the result of Table 1.

Method	Throughput(Gbps)
Proposed EG	70
RP-8stages	37
CP-8stages	60

implemented with Verilog language. We select Xilinx ISE 8.2i as the simulator. The device we choose to is "Xilinx Virtex-II Pro FPGA device XC2VP70" [10].

Table 1 shows the comparison of our scheme and the other two schemes in the three main categories, including throughput, memory usage, and Slice. We random select 30K prefix in [3] as our tested database to simulate. Both the three scheme can achieve the throughput at OC-768. Our proposed scheme get better throughput than the other two schemes. As in the memory domain, our scheme requires 1.25Mb due to the reason that we only use 37 bits for a prefix and 32 bits for range key. Ring Pipeline and Circular pipeline require 3.49Mb and 3.88Mb respectively to store the tree node. Our scheme does not scale well in the domain of slice. The reason is that lots of comparators are used for both range comparators and match comparators. The encoding step also requires hardware cost. The other two schemes scale well in the slice because they both are data-level pipeline.

Table 2 is the conjecture based on the simulation of original multi-bit trie and the result of Table 1. As we can see, the throughput of the other two schemes will decrease while the data structure transformed to multi-bit trie. The update overhead will occur as mentioned above. Figure 8 is the analysis when 1000 update operations take place in two tables. As the figure shows, most of the update operation will interrupt once in the proposed scheme. In the other hand, 8-stage multi-bit trie style will need 1.2 to 1.4 interrupts to complete the update operation.

5. Conclusion

In the paper, we proposed a pipelined IP forwarding scheme which aim on preventing the update overhead and improve the throughput. We first partition the prefix into disjoint groups based on the cover and covered relation of the prefix. We further partition these disjoint groups into smaller subgroups by the range key. We interleave store the prefixes in the same subgroup into different memory module to achieve faster search speed. With the experiment results above, two of the proposed schemes can achieve the throughput at OC-768. The update overhead can also be reduced.

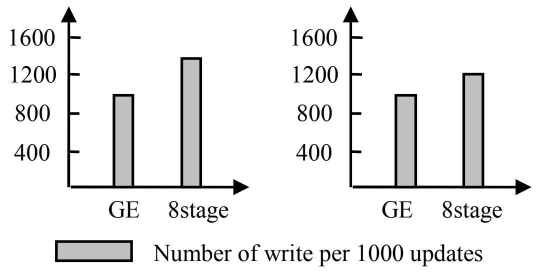

Figure 8. Analysis 1000 update operations.

References

[1] Florin Baboescu, Dean M. Tullsen, Grigore Rosu, Sumeet Singh, "A tree based router search engine architecture with single port memories," *Proceedings of 32nd International Symposium on Computer Architecture*, pp. 123-133, 2005

[2] Anindya Basu and Girija Narlikar, "Fast incremental updates for pipelined forwarding engines," *IEEE/ACM Transactions on Networking*. vol. 13, no. 3, pp.690-703, June 2005

[3] BGP Routing Table Analysis Reports, http://bgp.potaroo.net/.

[4] Yeim-Kuan Chang and Yung-Chieh Lin, "Dynamic Routing Tables Using Simple Balanced Search Trees", *Lecture Notes on Computer Science*, vol. 3961, January 2006.

[5] CACTI. http://quid.hpl.hp.com:9081/cacti/.

[6] M.Gray. Internet Growth Summary, http://www.mit.edu/people/mkgray/net/internet-growth-summary.html, 1996

[7] Sailesh Kumar, Michela Becchi, Patrick Crowley, and Jonathan Turner, "CAMP: fast and efficient IP lookup architecture", Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems, pp. 51-60, 2006.

[8] Stefan Nilsson and Gunnar Karlsson, "IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp.1083-1092, June 1999.

[9] Tammel, A. "How to Survive as an ISP," *Proceedings of Networld* (Interop '97).

[10] Xilinx Virtex-II Pro FPGA. http://www.xilinx.com