Simultaneous Multithreading-Based Routers

Kemathat Vibhatavanij†, Nian-Feng Tzeng†, and Angkul Kongmunvattana‡
†Center for Advanced Computer Studies, University of Louisiana at Lafayette
‡Department of computer science, University of Nevada
{kxv6043, tzeng}@cacs.louisiana.edu, angkul@cs.unr.edu

Abstract

This work considers the use of an SMT (simultaneous multithreading) processor in lieu of the conventional processor(s) in a router and evaluates quantitatively the potential gains as a result. An SMT processor exploits the benefits of both ILP (instruction level parallelism) and TLP (thread-level parallelism), suitable for the next generation routers, in which an increased number of functions are to be implemented. The use of an SMT processor not only allows router functions to be decomposed into multiple threads but also designates separate threads to handle different incoming traffic streams of a router to exploit TLP, potentially attaining performance improvement. Additionally, an SMT processor may admit new router functions or added traffic streams relatively easily without compromising much existing performance levels, via including a new thread (or threads) to perform one newly added function or traffic stream. This router design appears to have better flexibility and adaptability. In order to assess the benefits of this design approach, we implemented three key router functions (i.e., packet header extraction, packet header manipulation, and longest-prefix matching) as threads using an SMT simulator (SMTSIM) for performance evaluation. The results of this router design approach are collected and compared with those of conventional routers.

1. Introduction

Routers can be designed in general following two types of system architectures: centralized and distributed ones. A major advantage of the centralized architecture is its data consistency. Since routing/forwarding information is kept in only one (central) place, there is no concern about updating information to cause data inconsistency. On the contrary, the distributed architecture employs multiple line cards, with each line card incorporating a dedicated processor to take care of packet forwarding. If the contents of the routing table and forwarding tables (located in the forwarding engines, which may be inside or outside line cards) are not consistent, routing loops or inefficiency may occur. With the distributed architecture, however, one can add more line cards to increase router performance.

Both types of router architectures usually employ conventional processors as their central processing units (CPUs). With a single CPU in the router, all tasks involved in the router have to be done by that CPU, including routing table creation, packet header extraction and manipulation, routing tables lookups, packet forwarding, among others. If multiple CPUs are incorporated in the router, it is often that some of those CPUs are designated for routing table creation and for system control and administration, while other CPUs may be housed in the line cards (or forwarding engines) to handle input/output queues management, packet header extraction and manipulation (or routing table lookups, packet forwarding), and so on.

The SMT (simultaneous multithreading) processor supports both instruction-level parallelism (ILP) and thread-level parallelism (TLP). Its architecture combines the hardware feature of wide-issuing superscalars with the latency-hiding ability of the multithreaded architecture [3]. In order to support ILP and TLP, the SMT architecture deviates itself from its ancestors by five main aspects [1]. First, the SMT processor has multiple program counters. Second, it has a separate return stack for each thread. Third, it has per thread instruction retirement, instruction queue flush and trap mechanisms. Fourth, each branch target buffer has a thread identification. Fifth, it has a large register file. The first three and the fifth aspects are aimed to keep track of the per-thread state, whereas the fourth one is to avoid predicting phantom branches. The details of SMT architecture can be found in [2], [4]. SMT supports TLP by allowing multiple threads to share its functional units simultaneously. Since SMT exploits both ILP and TLP, it has a potential to achieve a higher throughput and a speedup when compared to its counterparts.

Designing the next-generation routers becomes a challenging task because the designer has to allow the addition of router's service functions without sacrificing too much performance of existing functions. A common design approach simply adds more hardware to increase processing power when new service functions arise. However, this would make the

0-7695-0768-9/00 $10.00

system more complicated and is often difficult to keep up with the new demands because of potential bottlenecks present in a conventional processor architecture.

Nowadays, various service functions are implemented inside a router, such as packet classification, packet forwarding, quality-of-service, queue management, and so on. These functions can be decomposed into threads to take advantage of the SMT architecture. Whenever a new function is added to the router, it can be implemented as a thread or threads. In order to demonstrate and evaluate the potential of applying the SMT processor in a router, we implemented three router functions, i.e., (1) header extraction, (2) header manipulation, and (3) longest-prefix matching, as separate threads or a single thread to handle one incoming traffic stream of a router. For comparison, we considered a wide-issuing superscalar processor that handles all the traffic streams of a router, as a counterpart (which is in contrast to the SMT processor where separate threads are employed to deal with different traffic streams). The measure of interest chosen for comparison is the number of packets processed by these two types of processors during a given time interval.

2. Pertinent Background

This section briefly describes background pertinent to the router functions implemented for performance evaluation, including packet header extraction and manipulation, and longest-prefix matching.

2.1 Header extraction and manipulation

A network architecture comprises a set of layers and protocols. A protocol is a collection of rules for communications in a layer, with its format usually composed of two parts: the header and the data. The header part is used for keeping control information and delimiting one layer from another, while the data part is the data portion, which can be viewed as a protocol data unit of the next higher layer. As our focus here is on IPv4 (Internet Protocol version 4), we shall explain the header of IPv4 in sequence, stating how it is manipulated when it passes from one router to the next. The header format of IPv4 consists of twelve common fields [11] as shown below.

When a packet arrives at a line card in the routers, its header is located and extracted [12], [13]. A part of the header is then transmitted to a forwarding engine for table lookups, including longest-prefix matching. The forwarding engine may be within or outside a line card. Once table lookups are done, the results are passed to the line card where the packet arrived earlier, in order to carry out header manipulation therein. Header manipulation mainly includes checking the protocol version, the header length, the TTL, and the header checksum. If TTL > 0 and the header checksum is correct, TTL is decreased by 1 and a new header checksum is calculated. The modified header is attached to the data part, which is kept in the same line card where the header manipulation takes place, to form a complete packet for delivery to the next router along the specified output port/interface card. If a packet has to be fragmented before delivery (because the maximum data unit in the next network segment is smaller), packet header manipulation then involves creating a new header for each data fragment, according to the original header. Additionally, it recalculates the offset of each fragment, the header checksum, masks, and the flags field. The fragmented packets are then sent off in sequence.

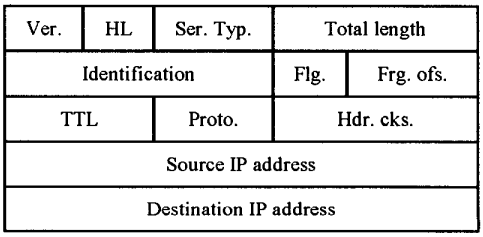


Fig. 1. Header format of an IP datagram.

2.2 Longest-prefix matching

A packet passes through a router from an incoming port interface to an outgoing port interface dictated by the destination address lookup result(s) in the maintained routing tables. Routing table lookups are non-trivial because a routing table records address prefixes (made possible with hierarchical addressing under IP or OSI), rather than complete address (of fixed length), to reduce the number of table entries. An address prefix denotes a group of contiguous addresses reachable through the same port interface of a router. The search of a given address in the routing table is rather complicated as multiple prefixes may match the address, and intuitively, the longest matched prefix should be chosen because it is the most specific one. This is thus referred to as longest prefix matching, with different algorithms introduced previously [17] for this purpose. The trie-based algorithms have been most popular, following various trie implementations, [6], [7], [8], [9]. In this study, we adopted the latest trie implementation known as dynamic prefix tries (DP-tries) [9] because of its simplicity and its available source code provided in [9], with some modifications made

to correct problems therein, as will be detailed in the next section.

3. Methodology

Our goal is to evaluate the potential performance gain resulting from the replacement of the conventional processor(s) with an SMT processor in the router. Three router functions are implemented using C, and they follow the IPv4 protocol outlined in the preceding section.

3.1 Header extraction and header manipulation functions

The header of an IP packet is represented as a structure data type, as depicted in Fig. 2, with one member corresponding to a field in the packet header. For example, the member of "ip_len" corresponds to the length field in an IP packet header.

The header extraction function first identifies an input IP packet and extracts the destination IP address. After the header of an IP packet is extracted and longest-prefix matching is carried out according to the destination field in the header, the header of the packet is modified before the packet is forwarded to the proper outgoing port/line card based on the matching result. This header modification includes updating the TTL field by 1 and recalculating the header checksum. If TTL equals 0, the packet is dropped. The checksum is obtained by fragmenting the header into 16-bit blocks and adding the blocks together via one's complement arithmetic operations; the final result is inverted before being put into the checksum field.

```
Struct ip {
    unsigned char   ip_verlen; /* IP ver. & hdr length */
    unsigned char   ip_tos;    /* type of serv. */
    unsigned short  ip_len;    /* total pkt length */
    short           ip_id;     /* datagram id */
    short           ip_fragoff;/* frg. offset */
    unsigned char   ip_ttl;    /* time to live */
    unsigned char   ip_proto;  /* IP protocol */
    short           ip_cksum;  /* hdr. checksum */
    unsigned int    ip_src;    /* source IP addr. */
    unsigned int    ip_dst;    /* dest. IP addr. */
        };
```

Fig. 2. Structure data type of the IPv4 header.

3.2 Longest-prefix matching function

Classless Inter-Domain Routing (or CIDR [5]) was introduced in order to scale the Internet routing table and slow down the exhaustion of the IP address space. According to the CIDR specification, an IP address can be split into network and host identifiers at any point, more flexible than the class-based structures (including classes A, B and C). For example, sixty-four consecutive class C addresses (202.10.0.0 - 202.10.63.0) under the class-based structures take up sixty-four entries in the routing table. On the contrary, those addresses under CIDR can be aggregated into a single network prefix with an explicit mask (202.10.0/18: a network prefix 202.10.0 with an 18-bit mask). Consequently, this classless network prefix and its mask (or prefix length) appear as a single entry in the routing table. However, CIDR poses the possibility that the same network prefix with different prefix lengths may appear as different table entries in a routing table. In order to determine a suitable entry for a given packet address, a lookup algorithm known as longest-prefix matching, is adopted (as opposed to an exact match in the routing table lookup).

A longest-prefix matching algorithm in the router attempts to find the best match between the network prefix entries in a routing table and the given packet address. After the best match is found, the router uses information kept in the entry to route the packet. As an instance, given two routing table entries: 24.64.128/18 and 24.64/10, and a packet address: 24.64.128.15, the best match entry for the packet address is 24.64.128/18, since it is more specific than the other entry. Therefore, the packet is forwarded to the output port specified by the routing table entry of 24.64.128/18.

Many IP lookup algorithms have been proposed [6], [7], [8], [9]. We selected to implement the longest-prefix matching algorithm published in [9], where a pseudo code was provided and well explained. In their algorithm, the routing table is kept in a dynamic prefix trie (DP-Trie) structure [9], with the node structure of the DP-Trie represented as shown in Fig. 3. Each node consists of one index and five pointers: parent, leftSubTrie, rightSubTrie, leftKey, and rightKey.

The algorithm is realized by three main operations on the DP-Trie: insertion, deletion, and retrieval [9]. The first operation is used for building a routing table and for entry insertion. The second one is used for entry deletion and restructuring the DP-Trie structure, while the third one is used for entry searching and for retrieving information from the searched entry. In our implementation, we modified the previous DP-Trie data structure by adding an interface pointer in the leftKey and the rightKey to keep their corresponding output interfaces, as illustrated in Fig. 4. Note that some part of the pseudo code given in [9] invoked wrong functions and passed improper arguments, and our implementation has corrected those errors.

- Index(n) represents the number of prefix bits that are common among all the keys in the subtrie that has node "n" as its root.
- Parent(n) is a pointer of node "n" that points to its root node. There is no parent for the topmost node.
- LeftSubTrie(n) is a pointer of node "n" that points to its left subtrie with key "k" such that k[index(n)] = 0. If node "n" has no left subtrie, it is a NULL pointer.
- RightSubTrie(n) is a pointer of node "n" that points to its right subtrie with key "k" such that k[index(n)] = 1. If node "n" has no right subtrie, it is a NULL pointer.
- LeftKey(n) is a key with leftKey[index(n)] = 0. If node "n" has no left key, it is a NULL pointer.
- RightKey(n) is a key with rightKey[index(n)] = 1. If node "n" has no right key, it is a NULL pointer.

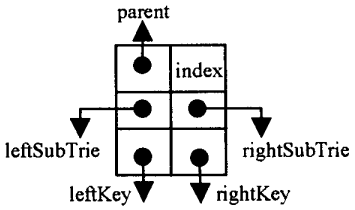


Fig. 3. A node structure of the DP-Trie.

Key/mask	interface
4/255	192.41.177.2
6/255	192.41.177.8
9.20/255.255.128	192.41.177.110
10.12.12/255.255.255	192.41.177.133
12/255	192.41.177.3

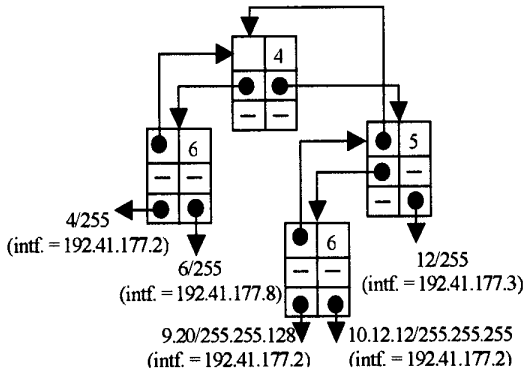


Fig. 4. DP-Trie structure in our implementation.

3.3 Simulator SMTSIM

SMTSIM is an emulation based, instruction-level simulator [2], written to run on a DEC Alpha machine. It executes unmodified Alpha object codes and models the execution pipelines, memory hierarchy, TLBs, and branch prediction logics of a wide-issuing superscalar. The simulation parameters employed in this work are as follows:

- fetch and decode bandwidth: 8 instructions per cycle
- 6 integer units and 4 out of the 6 units can execute load/store instructions
- 3 floating point units
- 100 integer renaming registers
- 100 floating point renaming registers
- integer instruction queue with 32 entries
- floating point instruction queue with 32 entries
- instruction TLB with 48 entries
- data TLB with 128 entries
- 64/256/1024 KB L1 instruction cache
- 64/256/1024 KB L1 data cache
- 1/2/8 MB L2 unified cache
- 16/64 MB L3 unified cache
- 64 bytes per block for all caches
- 2K×2-bit PHT (prediction history table)
- BTB (branch target buffer) with 256 entries
- maximum number of threads: 8.

3.4 Synchronization among threads

In our simulation, three threads are implemented and executed in SMTSIM, with one thread responsible for packet header extraction, one for packet header manipulation, and the third one for longest-prefix matching. Note that the three threads for any input stream of a router are not totally independent. For example, the header extraction thread has to finish before advancing to the longest-prefix matching thread, which in turn, is followed by the header manipulation thread. Synchronization among these threads for each input stream is thus necessary. Any desirable scheme for SMT thread synchronization should yield high performance and deadlock freeness, while consuming as little resources as possible. A recent paper introduced two primitives, `Acquire(lock)` and `Release(lock)`, for this synchronization purpose and showed their effectiveness [16]. In our implementation, a different synchronization scheme was attempted, as explained below.

Our thread synchronization is realized by means of thread communication in the "commit.c" module, which represents the last pipeline stage of an SMT processor. Any instruction reaching this stage has been executed properly, with corresponding registers and memory locations updated as well. At that point, there is no wrong path execution, and it is therefore safe to carry out thread communications. Two arrays of an equal size are allocated for communications between two dependent threads (e.g., the header extraction and the longest-prefix matching threads), with one array in one thread. No thread has an access to the array in the other thread. Data accesses in an array are managed by two indexes: the head and the tail indexes, which are kept at the

end of each array. SMTSIM can access both arrays and uses two indexes to keep track of reading and writing positions in the two arrays. For explanation, the arrays and their indexes are named as given in Fig. 5. Arrays are implemented to function as circular queues, with provision for differentiating between an empty queue and a full queue. To this end, a technique called "empty position" is employed to prevent data overwrites and to indicate whether a queue is empty or not, with steps involved shown below.

```
tailIndex = (tailIndex+1) % sizeOfArray;
if (tailIndex != headIndex)
 {update the previous entry;}
 else
 {tailIndex--;} /* queue is full */
```

Initially, the head index (hx_h) and the tail index (hx_t) point to the same position. After a header extraction thread extracts the destination IP address from a packet header, it writes the address into its queue (i.e., hx_queue), provided that the queue is not full. In every commit cycle, SMTSIM checks both queues (i.e., hx_queue and lpm_queue) to see whether there is any unread data in queue hx_queue and also whether queue lpm_queue is available, if hx_queue has unread data in it. SMTSIM then moves the unread data, if any, from hx_queue to lpm_queue, before advancing smt_h (lpm_t) and smt_t (hx_h) by 1 using the modulo number operation. The longest-prefix matching thread reads any data put in its queue (i.e., lpm_queue).

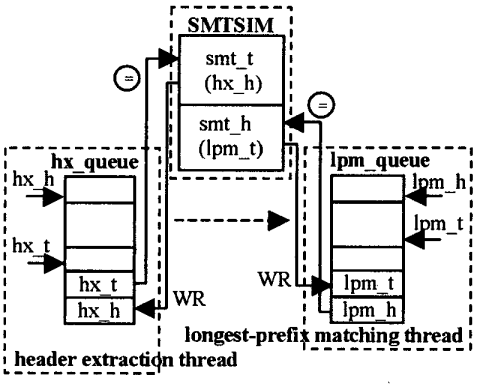


Fig. 5. Synchronization between two threads.

Synchronization between the longest prefix matching thread and the header manipulation thread follows the same scheme as described above.

4. Simulation Setting and Results

A larger register file exists in SMT, and thus longer access times result [14]. SMTSIM [2] simulates the SMT pipeline which is two cycles longer than the pipeline of a compatible superscalar processor (e.g., MIPS R10000 [15]) to keep the cycle time unchanged. It might be expected that a single thread executing on an SMT processor yields longer latencies and lower performance. However, it has been shown [4] that when compared with a superscalar processor, SMT with a single thread exhibits the performance level of no more than 1.5% lower than that of a single-threaded superscalar (like MIPS R10000). In this comparative simulation study, the pipeline is considered to have identical performance for SMT with a single thread and for a compatible superscalar processor. The SMT processor is evaluated with three threads (for header extraction, header manipulation, and longest-prefix matching) existing to deal with each incoming traffic stream at a port/interface card of the router. Since the three threads are not independent and require synchronization among them, we also simulated the scenario where a "combined thread" formed by putting the three threads together is employed to handle one traffic stream of the router. This scenario arrives at a performance upper bound when synchronization overhead is absent. For comparison, a router with one conventional superscalar processor to handle all traffic streams reaching its ports/interface cards was assessed as well. Each simulated router configuration runs for 50 million clock cycles under SMTSIM; at the end of each simulation run, the total number of packets processed (namely, packet throughput) under the simulated configuration is gathered as the performance measure of interest.

In our simulation, a stream of input packets is simulated by reading an IP traffic file, which we created by capturing IP packet traffic in a fast Ethernet connected to one port of a Cisco 7000 router. One IP traffic file was created for a router port/interface card. Such a traffic file recorded traffic patterns on a port of an edge router rather than a backbone router. However, any conclusion drawn from the simulation results obtained using this type of traffic files for an edge router is believed to hold true for a backbone router as well. The routing table is created using the Mae East database [10], and it is created before the simulation starts to process any packet stream.

SMTSIM is run on our Alpha workstation, AlphaStation XP 1000, which is equipped with an Alpha 21264A 667 MHz processor, 64 KB data cache and 64 KB instruction cache, 4 MB L2 cache, and 128 MB ECC memory, and which runs the Compaq Tru64 UNIX operating systems.

4.1 Simulation Results

Packet throughput is the measure of interest, and it depends largely on the L1 cache organizations of the SMT processor. In our simulation study, the parameters of cache organizations are chosen as listed in Table 1. Recall that the L1 cache is split for instructions and data, while L2 and L3 are unified caches as indicated in Sec. 3.3.

Table 1. Parameters of cache organizations (C. org.) chosen for our simulation study

C. org.	L1(KB)	L2(KB)	L3(MB)	Asso.deg.
64K/2	64	1024	16	2
64K/4	64	1024	16	4
256K/2	256	2048	16	2
256K/4	256	2048	16	4
256K/8	256	2048	16	8
1M/2	1024	8192	64	2
1M/4	1024	8192	64	4
1M/8	1024	8192	64	8

Packet throughput as a function of L1 cache organizations in SMT for a router with two traffic streams is depicted in Fig. 6. As can be found from the results of employing three threads to handle each traffic stream in this figure, no noticeably higher throughput results from increasing the L1 cache size to 256 KB or beyond (say, to 1 MB) from 64 KB. This is probably because the L1 cache with size 64 KB is large enough to accommodate the working set under the associativity degree equal to 2 or higher.

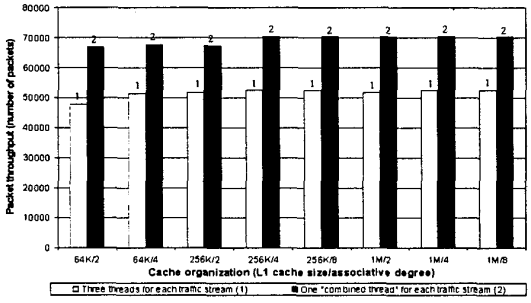


Fig. 6. Packet throughput under different L1 cache org. (with two traffic streams reaching the router).

To explore the impact of thread dependency on SMT performance, we put the three threads together as one "combined thread" for dealing with a traffic stream of the router and collected throughput results under the same set of cache parameters for comparison, as demonstrated in Fig. 6. As somewhat expected, the use of combined threads in the SMT processor eliminates the communication overhead existing among threads, hence leading to somewhat higher throughput. For example, when the L1 cache is of size 256 KB with the associativity degree being 4, combined threads attain a throughput gain of about 30%. Dependency among threads in the SMT processor for this router application makes it favorable to implement coarse-grained threads (as our "combined threads"), with one such coarse-grained thread for one traffic stream and multiple such coarse-grained threads for handling those concurrent traffic streams at a router. Note that the number of such threads in SMT is not necessarily equal to the number of ports/interface cards of the router. For a given router, an optimal number of simultaneous threads exists and the number depends mainly on the L1 cache organization in the SMT processor, as will be explored subsequently. From this point onward, we shall present the throughput performance of an SMT processor with combined threads only.

The packet throughput of a router with eight ports/interface cards handled by an SMT processor was measured by varying the number of threads, the degree of associativity, and the cache size. As shown in Fig. 7, the packet throughput grows consistently as the number of (combined) threads grows from 1 to 4, for all L1 cache organizations with the associativity degree of 4 or 8. For the associativity degree 2, however, the packet throughput peaks with the number of threads equal to 2, suggesting that such an associativity degree cannot support more than two threads no matter how large the cache size is. When the number of threads goes beyond 4, the throughput level starts to drop as the number of threads grows further, unless the degree of associativity in L1 is increased accordingly from 4 to 8. If the associativity degree is 8, the packet throughput moves up slightly before leveling off when the number of threads grows beyond 4, for the L1 cache size of 256 KB or 1 MB. In addition, it is found that the L1 cache size of 256 KB is large enough for up to 8, and expanding the cache size beyond 256 KB has little positive impact on packet throughput.

When its L1 cache size is chosen, an SMT processor in this router application delivers unsatisfactorily low performance if the degree of associativity is not large enough to accommodate the simultaneous threads present in the SMT processor, leading to contention in set entries. In other words, SMT performance is sensitive to the degree of associativity, and an increased number of threads requires a correspondingly larger associativity degree. On the other hand, for a given L1 cache size and associa-

tivity degree, SMT exhibits an optimal number of threads. As an instance, for L1 of size 256 KB and the associativity degree 4 (or 8), the optimal number of threads is 4 (or 5), according to Fig. 7.

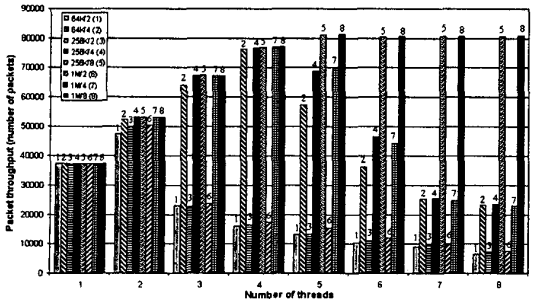
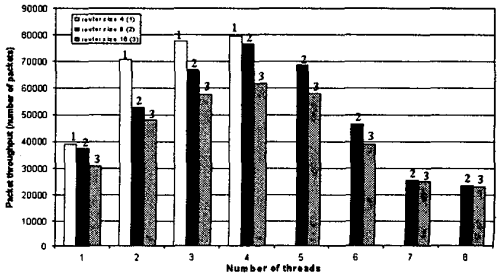


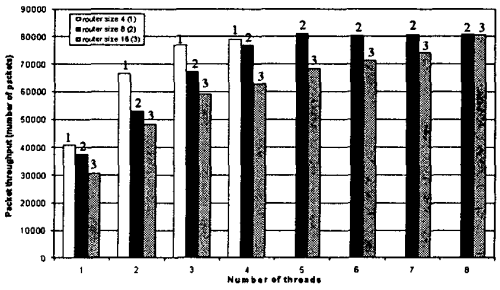
Fig. 7. Packet throughput versus number of threads under different L1 cache org. for a router with eight (8) ports/intface cards.

In order to compare SMT with a compatible superscalar processor in this router application, we assume that the superscalar processor has the same cache organization and the same set of resources (e.g., integer units, floating point units, instruction queues, etc., as listed in Section 3.3) as SMT under comparison. We further assume that there is one and only one thread existing in the superscalar processor to handle a traffic stream. In other words, we effectively ignore the overhead involved for the superscalar processor to switch processing from one traffic stream to another in the router, at which multiple traffic streams arrive; the actual performance level of the superscalar processor is upper bounded by the results obtained in this way. The performance of a compatible superscalar processor under these assumptions is depicted in Fig. 7 under the case of one thread (i.e., the leftmost group). According to the results shown, a superscalar processor is insensitive to the L1 cache organization, since it delivers almost identical performance for all cache organizations. An L1 cache of size 64 KB is sufficient for the superscalar processor. This is because of our assumption that the same thread is employed to handle all incoming traffic streams in the router without switching contexts, rendering the working set far smaller than that of a compatible SMT processor (with multiple threads) and thus contention in set entries unlikely. This smaller and simpler L1 cache requirement for the superscalar processor results in its lower packet throughput than SMT, since it fails to explore TLP (thread-level parallelism). We may use the performance level of a compatible superscalar processor as a basis for measuring the performance gains using SMT with different L1 cache organizations. Given a router with 8 incoming traffic streams, for example, it is observed from Fig. 7 that SMT with 4 threads can achieve a performance gain of more than 100% when compared with a compatible superscalar processor, when the L1 cache has an associativity degree of 4 or beyond. The performance gain for 2 threads under the associativity degree 2 is about 25% for the L1 cache size of 64 KB.

It is also interesting to find out the optimal number of threads in SMT applied to different sized routers, for a given L1 cache organization. Packet throughput versus the number of threads for different router sizes are illustrated in Fig. 8(a), where the L1 cache size equals 256 KB and the associativity degree is 4, with L2 and L3 being 2 MB and 16 MB, respectively. The optimal number of threads for a router sized 4 is found to be 4 from this figure, under all the router sizes considered. If the associativity degree is 8, the optimal thread number is 5 (or 8) for a router sized 8 (or 16), according to Fig. 8(b). Given an L1 cache organization, the optimal number of thread varies for different router sizes.



(a) L1 cache org. = 256KB, associative deg. = 4.



(b) L1 cache org. = 256 KB, associative deg. = 8.

Fig. 8. Packet throughput versus number of threads for different router sizes under given L1 cache org.

5. Conclusions

This paper has explored the application of an SMT processor to the routers and assessed its potential performance benefits when compared to a

conventional processor with identical resources (e.g., instruction fetch and decoding logics, integer and floating-point units, renaming registers, and so on), using an SMT simulator called SMTSIM [2]. Three key router functions were implemented (i.e., header extraction, header manipulation, and longest-prefix matching), and they are treated as threads in SMT. However, these threads are dependent, and synchronization among them is necessary, causing some performance degradation when compared with the implementation without synchronization made possible by putting the three threads together as one "combined thread." SMT allows multiple (combined) threads to co-exist for dealing with multiple incoming traffic streams of a router. The traffic streams employed in our simulation study were obtained by capturing IP packet traffic in a fast Ethernet connected to one port of a Cisco 7000 router. The routing table is created using the Mae East database [10], and it is created before the simulation starts to process any packet stream.

The simulation results demonstrate the advantage of SMT by exploiting thread level parallelism (TLP), in addition to instruction level parallelism. For the router application, SMT may yield a performance gain (in terms of packet throughput) of more than 100% resulting from TLP exploitation. It is found that the L1 cache organization, dictated by its size and associativity degree, for SMT processors has a substantial impact on the performance gain. To maintain good performance, the cache associativity degree has to be large enough to accommodate the working set of all threads simultaneously existing in SMT, avoiding contention in the set entries. The application of SMT to routers offers a potential in handling considerably more packets during a unit time span and is clearly a preferred approach to high-performance router design.

6. References

[1] D. M. Tullsen *et al.*, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proc. of the 22nd Annual Int'l Symp. on Comp. Arch.*, Santa Margherita Lingure, Italy, June 1995, pp. 392-403.

[2] D. M. Tullsen *et al.*, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreaded Processor," *Proc. of the 23rd Annual Int'l Symposium on Computer Architecture*, Philadelphia, PA, May 1996, pp. 191-202.

[3] R. Alverson *et al.*, "The Tera Computer System," *Int'l Conf. on Supercomputing*, June 1990, pp. 1-6.

[4] S. J. Eggers *et al.*, "Simultaneous Multithreading: A platform for Next-Generation Processors," *IEEE Micro Magazine*, vol. 17, no. 5, pp. 12-19, September/October 1997.

[5] V. Fuller *et. al.*, "Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy," *Internet Engineering Task Force*, RFC (Request for Comments) 1519, September 1993.

[6] B. Lampson *et. al.*, "IP Lookups Using Multiway and Multicolumn Search," *IEEE/ACM Transactions on Networking*, vol.7, no. 3, pp. 324-334, June 1999.

[7] S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Tries," *IEEE Journal on Selected Areas in Comm.*, vol. 17, no. 6, pp. 1083-1092, June 1999.

[8] N.F. Huang and S.M. Zhao, "A Novel IP-Routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1093-1104, June 1999.

[9] W. Doeringer *et al.*, "Routing on Longest-Matching Prefixes," *IEEE/ACM Transactions on Networking*, vol. 4, no. 1, pp. 86-97, February 1996.

[10] URL: http://www.merit.edu/ipma/routing_table

[11] J. Postel, "Internet Protocol," *Internet Engineering Task Force*, RFC (Request for Comments) 791, September 1981.

[12] C. Partridge *et al.*, "A 50-Gb/s IP Router," *IEEE/ACM Transactions on Networking*, vol. 6, no. 3, pp. 237-247, June 1998.

[13] S. Keshav and R. Sharma, "Issue and Trends in Router Design," *IEEE Communications Magazine*, vol. 36, no. 5, pp. 144-151, May 1998.

[14] J. L. Lo *et al.*, "Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 322-354, Aug. 1997.

[15] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro Magazine*, vol. 16, no. 2, pp. 28-40, Apr. 1996.

[16] D. M. Tullsen *et al.*, "Supporting Fine-Grained Synchronization on a Simultaneous Multithreading Processor," *Proc. 5th Int'l Symp. on High-Performance Comp. Arch.*, Jan. 1999, pp. 54-58.

[17] H. Tzeng and T. Przygienda, "On Fast Address-Lookup Algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1067-1082, June 1999.