Evaluation of the content-based packet scheduling policies on the multithreaded multiprocessor network system

Kangbin Yim
Dept. of Information Security Engineering, Soonchunhyang University, Asan, 336-745 Korea
Tel : +82-41-530-1741 Fax : +82-41-530-1639 E-mail: yim@sch.ac.kr

Abstract: In this paper, I propose a thread scheduling policy for faster packet processing on the network processors with multithreaded multiprocessor architecture. To implement the proposed policy, I derived several basic parameters related to the thread scheduling and included a new parameter representing the packet contents and the features of the multithreaded architecture. Through the empirical study using a network processor, I proved the proposed scheduling ploicy provides better throughput and load balancing compared to the generally used thread scheduling policy.

Keywords: Processor, Multiprocessor, Multithreaded, Thread Scheduling

1. INTRODUCTION

According to the drastic increments of the Internet speed and the number of its users various network protocols and the related services recently have been introduced, which causes existing network equipments to be more powerful and more adaptive. For network equipments to provide more powerful performance, it is essential to process the traffic using ASIC-based hardware platforms. However, this approach rises up a serious drawback, lack of adaptability over the progressively ongoing protocol environment. Therefore, most systems have not been able to get out of using general processors for the flexibility of the software upgrades at the cost of performance degradation. However, the general processor based traffic processing has got stuck with the performance limitation as giga-bit network environment is going to take its place. Because of the reason many recent researches[1][2][3][10] are focused on the specially designed network processor that is both flexible to support new services and capable to process high speed packet traffic.

General network processors implement the lowest part of the packet processing task in hardware and leave the rest part to be organized on the packet processing element. As a high performance RISC processor, the packet processing element is able to support new coming services by accommodating new or modified program[10]. Moreover, a single network processor can embed multiple packet processing elements for more powerful packet processing each of which is implemented based on the multithreaded architecture capable of supporting hardware level context switching. Therefore, one single network processor supports the number of threads equal to the number of packet processing elements multiplied by the number of threads per packet processing element.

The throughput of the packet processing service implemented on the network processor is likely to partly depend on the assignment of the functions to tasks and the arrangement of the tasks to the threads. This means that the performance of the packet processing system exceedingly depends on the scheduling policy for the threads processing the ingress packets.

Various researches on the scheduling algorithms have been done in the multiprocessor environment for efficient assignment of the tasks to processors. As an example, Parallel Loop Scheduling[5] divides a job into several loop containing processes and assigns each of them to each processor. This kind of scheduling policies can be categorized into two classes such as static scheduling and dynamic scheduling. Static scheduling gives low overhead but faces lack of load balancing and dynamic scheduling is vise versa. Other proposed dynamic scheduling algorithms including MSS[7], Pfairness[6] and more have been trying to take better position on the trade-off between the overhead and the load balance. There have been more scheduling algorithms than above, though most of them considered only general purpose jobs ignoring the characteristics of the packet processing tasks for the high speed network processor.

This paper proposes a thread scheduling policy that considers the characteristics of the multithreaded multiprocessor (MTMP) architecture of the network processor as well as the characteristics of the packet processing tasks on the network processors. The proposed policy is implemented and evaluated on the IXP1200[12] system that is cheap but incorporates most characteristics of generic architecture of the network processor.

In this paper, various thread scheduling policies utilized for network filtering systems are evaluated, including Round-Robin, FCFS and the proposed FCFS with SOP balancing algorithm, last of which may solve two drawbacks of the previous two: thread blocking problem in Round-Robin and SOP concentration problem in FCFS. The main idea of the proposed scheduling policy is that the processing workloads are different for different parts of a packet on most network systems such as the firewall and the intrusion detection system that incorporate the packet filtering task. According to the result of the empirical study the proposed policy gives gigabit performance for more than 1024 bytes in size of a packet.

2. PACKET PROCESSING ON THE NETWORK PROCESSOR

Designing of the packet processing software for MTMP based network processor needs to consider its physical features especially including the limitations of packet processing time, the characteristics of the inter-process

communication mechanism, the restrictions on register usage and so on. To maximize the packet processing throughput on the MTMP architecture it is required for scheduler to consider the workload of each thread as one of the scheduling parameters. Therefore how the thread's workload represented is very important. Because the workload of a thread depends on the type and size of the packet they should affect the expression of the workload.

Figure 1 simplifies the flow of the ingress packets inside the MTMP system. The size of the packet flowing into the input queue varies according to the protocol type or the size of a datagram on the data link layer (an L2 packet) and is independent of the unit size of the data processed by the threads on the processing elements. Generally, the unit size of the data on general network processors, called the MAC packet in this paper, is fixed and much smaller than the size of the L2 packet.

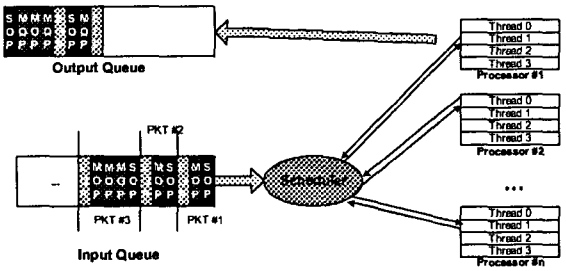

Fig. 1. Simplified packet processing on an MTMP system

Because of the reason above a single L2 packet is likely to consist of multiple MAC packets, and they can be classified into three different types such as SOP (Start Of a Packet), MOP (Middle Of a Packet) and EOP (End Of a Packet) according to their location within an L2 packet. The SOP, as a leading-end MAC packet, generally takes the header of the L2 packet and causes the processing threads heavier workload than other two. The MOP, as named, takes one of the interior partitions of an L2 packet and causes a simple block transfer and least workload except for the uncommon special applications. The EOP, as a trailing-end MAC packet, is related to the execution of the action from the decision of reject or forward. Therefore, the workload relationship can be shown as SOP>EOP>MOP and be represented as the effective parameters for an efficient scheduling algorithm.

3. SCHEDULING POLICIES

The requirements for efficient packet processing on the MTMP environment basically include minimum thread contention delay, and sufficient thread workload distribution for maximum processor utilization or packet throughput. In this paper three scheduling policies are defined according to the way how to deal with the state and the load of each thread: static scheduling, minimum load scheduling and predictive minimum load scheduling. The scheduling policies are described in this section with definition. The symbols used in the definitions are listed on the table 1.

Static scheduling is the simplest scheduling policy that uses fixed order for selecting threads for MAC packets and provides lowest scheduling overhead. Because the order is fixed the entire processing tasks could be stalled by only a single busy thread. The busy thread here means that the thread has heavier workload than others. Therefore this policy can be efficient in case the threads' workloads are all the same. However, the workloads are different for different MAC packets especially for nowadays' complex services.

- Minimum load scheduling gives a new MAC packet to the thread that has currently minimum processing load. The information about the load is derived from the thread's state and the MAC packet type. To estimate the total load of a processing element each thread is checked for its load. Scheduler then selects the minimum loaded processing element and assigns the new MAC packet to one of the threads that may currently have minimum load on the processing element. All workload would be evenly distributed by this approach and maximum throughput could be achieved. However, this method has a drawback that possibility that multiple threads could have the same task on the same processing element would be increased, which rises up the overhead of the processing element. The scheduler is defined as the following.

$$S(t,\Pi_{at}) = u_j^i \quad where \quad u_j^i = \arg\min\{L_u(u_m) : u_m \in \pi_i\}$$
$$\pi_i = \begin{cases} \arg\min\{L_\pi(\pi_n) : \pi_n \subset \Pi\} & when \quad \Pi_{at} = \{\varnothing\} \\ \arg\min\{L_\pi(\pi_n) : \pi_n \subset \Pi_{at}\} & when \quad \Pi_{at} \neq \{\varnothing\} \end{cases}$$
$$L_\pi(\pi_i) = \sum_{j=0}^{N} L_u(u_j^i)$$

- Predictive minimum load scheduling assigns a new MAC packet to the minimum loaded thread as in the minimum load scheduling except that it minimizes the possibility to select one of the threads on the same processing element. In this method, a scheduling parameter α is defined to penalize for the overlaying the same task on the same processing element as shown bellow.

$$S(t,\Pi_{at}) = u_j^i \quad where \quad u_j^i = \arg\min\{L_u(u_m) : u_m \in \pi_i\}$$
$$\pi_i = \begin{cases} \arg\min\{L_\pi(\pi_n) : \pi_n \subset \Pi\} & when \quad \Pi_{at} = \{\varnothing\} \\ \arg\min\{L_\pi(\pi_n) : \pi_n \subset \Pi_{at}\} & when \quad \Pi_{at} \neq \{\varnothing\} \end{cases}$$
$$L_\pi(\pi_i) = \sum_{j=0}^{N} L_u(u_j^i) + d(\pi_i) \times \alpha$$

Table 1. Symbols used in the definitions

	symbol	description
PE π	Π	Set of PEs ($\bigcup_{i=0}^{N} \pi_i \in \Pi$)
	Π_{at}	Set of PEs having more than one idle thread at time t
	π_i	i th PE
	$L_\pi(\pi_i)$	Load of the i th PE
	$d(\pi_i)$	If the previous PE was π_i that run the scheduler
Thread u	u_j^i	j th thread on i th PE ($\bigcup_{j=0}^{N} u_j^i \in \pi_i$)
	$L_u(u_j^i)$	Load of the u_j^i
Scheduler	α	Overhead of overlaying

4. EMPIRICAL RESULTS

The proposed scheduler was implemented on the Intel IXP1200 that has a main processor and six processing elements each of which has up to four hardware threads. In this paper two processing elements were assigned for each gigabit network interface. Packet processing throughput was measured for various packet sizes. For each test on the same size of packets, only same sized packets were generated and fed to the network system to watch the performance differences more vividly.

Figure 2 shows the results. In the figure, SS, ML and MLP shows the results of static scheduling, minimum load scheduling and predictive minimum load scheduling, respectively. As I expected, the minimum load scheduling and predictive minimum load scheduling provides better performances compared to the static scheduling especially for the packets ranging larger than 256bytes in size. For the packets smaller than 256bytes, the performance results of the two policies are not so much better than those of the static scheduling because smaller packets are possibly to have a large percentage of header fields and the consideration of the packet content is not so effective.

In the figure, it is found that the static scheduling does not show the monotonic increase proportional to the packet size. Instead, it provides worse performance for the packets twice the MAC packet in size because the system implemented uses only two processing elements.

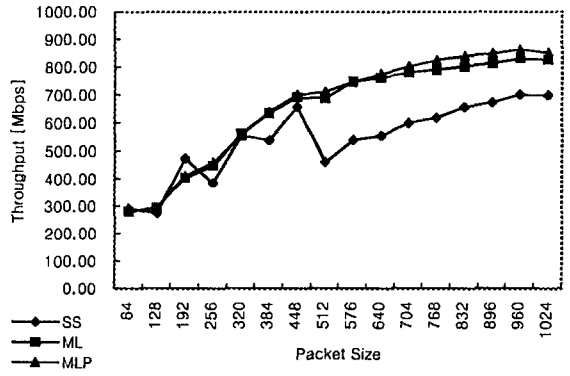

Fig. 2. Packet processing throughput for schedulers

5. CONCLUSION

This paper proposed a scheduling policy for the multithreaded multiprocessor based network processor system to maximize packet processing throughput by distributing the packet processing load based on the attribute of the packet content. To verify the performance of the proposed policy it was implemented on a real network processor with multithreaded multiprocessor architecture. The empirical results of the implementation show that the proposed policy provides better performance than the existing policies especially for larger packets.

Acknowledgement: This paper is the result of the work supported by 2004 Soonchunhyang University Research Fund.

References

[1] A. Campbell, H. De Meet, M. Kounavis, K. Miki, J. Vicente, and D. Villela, "A survey of programmable networks," ACM Computer Communications Review, Apr. 1999.

[2] P. Crowley, M. E. Fiuczynski, J. L. Baer, and B. N. Bershad, "Characterizing processor architectures for programmable network interfaces," Proc. of the International Conference on Supercomputing, 2000.

[3] Niraj Shah, Kurt Keutzer, "Network processors: Origin of species," Proc. of the Seventeenth International Symposium on Computer and Information Sciences, Oct. 2002.

[4] T. Wolf and M. A. Franklin, "Locality-aware predictive scheduling for network processors," Proc. of IEEE International Symposium on Performance Analysis of Systems and Software, Tucson, AZ, Nov. 2001.

[5] D.J. Lilja, "Exploiting the parallelism available in loops," IEEE trans. on computer, volume 27, no. 2, pp. 13-26, 1994.

[6] S. Baruah, J. Gehrke, and C.G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," Proc. Of the 9th International Parallel Processing Symposium, pp. 280-288, Apr. 1995.

[7] K. P. Hung, N. H. C. Yung, and Y. S. Cheung, "Multithreaded self-scheduling: Application of multithreading on loop scheduling for distributed shared memory multiprocessor," IEEE International Conference on Algorithms and Architectures for Parallel Processing, Brisbane, Australia, Apr. 1995.

[8] C. Polychronopoulos and D. Kuck, "Guided self-scheduling: Practical scheduling scheme for parallel supercomputers," IEEE Trans. on Computers, Vol. C-36, No. 12, pp. 1425-1439, Dec. 1987.

[9] Ten H. Tzen and Lionel M. Ni, "Trapezoid self-scheduling: Practical scheduling scheme for parallel supercomputers," IEEE Trans. on Parallel and Distributed Systems, Vol. 4, No. 1, pp. 87-98, Jan. 1993.

[10] Nie, X., Gazsi, L., Engel, F., Fettweis, G. "A new network processor architecture for high-speed communications," Proc. of IEEE Workshop on Signal Processing Systems, Taipei / Taiwan, Oct. 1999.

[11] V. Srinivasan and G. Varghese and S. Suri and M. Waldvogel, "Fast and scalable layer four switching," Proc. of ACM SIGCOMM, pp. 191-202, 1998

[12] Intel IXP1200 Network Processor Family, Hardware Reference Manual, Intel Corp. Dec. 2001