Journal of KIIT. Vol. 17, No. 2, pp. 79-85, Feb. 28, 2019. pISSN 1598-8619, eISSN 2093-7571 79
http://dx.doi.org/10.14801/jkiit.2019.17.2.79

A Sequence-Preserving Packet Scheduler for Multi-Core Network Processors

Seung-Ho Ok*, Byungin Moon**

This work was supported by 2018 Dong-eui University Grant (grant number 201802880001).

Abstract

Recently, network processors (NPs) use multiple packet processing elements (PEs) to exploit packet-level parallelism for high packet processing throughput. This paper presents a new sequence-preserving packet scheduler for a NP with multiple packet PEs. Even though packets of the same flow are processed in parallel by different PEs, the proposed scheduler preserves the correct output packet sequence of each flow by utilizing pre-estimated packet processing time. Experimental results show that the proposed scheduler is able to increase the NP throughput when compared with the conventional per-flow queuing (PFQ) method and round-robin (RR) packet scheduler. In addition, the proposed scheduler achieves this performance improvement without a large hardware overhead, in contrast to the conventional sequence number matching (SNM) method and the RR packet scheduler.

요 약

최근 네트워크 프로세서는 패킷 처리량을 높이기 위해 다중 패킷 처리기를 사용하여 패킷 수준의 병렬 처리를 수행한다. 본 논문에서는 다중 패킷 처리기를 사용하는 네트워크 프로세서를 위한 새로운 시퀀스 보존 패킷 스케줄러를 제시한다. 동일한 흐름의 패킷들이 상이한 패킷 처리기에 의해 병렬로 처리되더라도, 제안된 스케줄러는 미리 추정된 패킷 처리 시간을 이용함으로써 각 흐름의 올바른 출력 패킷 시퀀스를 보존한다. 실험 결과는 제안된 스케줄러가 기존의 PFQ(Per-Flow Queuing) 방법과 RR(Round-Robin) 패킷 스케줄러와 비교할 때 네트워크 프로세서 처리량을 증가시킬 수 있었다. 또한 제안된 스케줄러는 기존의 SNM(Sequence Number Matching) 방법과 RR 패킷 스케줄러와 달리 하드웨어 오버헤드 없이 성능을 향상시킬 수 있다.

Keywords
sequence-preserving packet scheduler, network processor, per-flow queuing, round-robin

* Assistatant Professor, Dept. of Robot and Automation Engineering, Dong-eui University
- ORCID: https://orcid.org/0000-0002-9036-0872
** Professor, School of Electronics Engineering, Kyungpook National University
- ORCID: http://orcid.org/0000-0002-8102-4818

· Received: Dec. 24, 2018, Revised: Jan. 28, 2019, Accepted: Jan. 31, 2019
· Corresponding Author: Byungin Moon
School of Electronics Engineering, Kyungpook National University,
80 Daehakro, Bukgu, Daegu 41566, Korea,
Tel.: +82-53-950-7580, Email: bihmoon@knu.ac.kr

Ⅰ. Introduction

Most commercially available NPs use multiple PEs to exploit packet-level parallelism for high packet processing throughput. As a result, packets that are processed by multiple PEs are likely to be transmitted out-of-order at the output, thereby leading to a severe degradation in network performance[1]-[4]. Thus, one of the most important requirements for an NP is the ability to preserve the correct output sequence of packets of the same flow.

Generally, the PFQ and SNM methods are used to preserve the sequence of the packets. In the case of the PFQ, incoming packets of the same flow are assigned to the same per-flow queue, and then dispatched in order to the same PE[5]. As each PE processes the packets assigned to it and sends them to the output link scheduler in the order that the packets come into the PE, the correct output sequence of packets of the same flow is easily maintained. However, when packets of the same flow come into the NP consecutively, the throughput of the NP deteriorates, as some PEs are in an idle state due to an insufficient number of active flows. Meanwhile, the SNM method assigns unique sequence numbers to incoming packets, which are then processed by multiple PEs regardless of the flow they belong to. As the packet processing times differ according to the features and lengths of the packets, the correct output sequence of packets of the same flow is then sorted using the packet sequence numbers and output buffers [6][7]. However, if a particular packet is delayed in a PE due to an unexpected exception, subsequent packets of the same flow also have to be delayed in the output buffers, thereby requiring output buffers with unpredictable sizes to sort packets of the same flow in the correct output sequence. As a result, the SNM method can involve excessive hardware costs to prevent an overflow of the output buffers.

On the other hand, in order to preserve the sequence of the packets in a packet scheduler, an RR based packet scheduler is proposed. In[8], a combined version of the surplus RR and deficit RR called packetized ordered round-robin (P-ORR) packet scheduling is proposed. Since this algorithm schedules given number of packets in each scheduling round, idle periods on PEs between adjacent scheduling rounds increase as the number of PEs increases. In addition, as the variation of the length of the packets increases, out-of-order packets are transmitted at the output.

Accordingly, to increase the packet throughput and PE utilization with minimal hardware overhead, we propose a sequence-preserving packet scheduler that exploits pre-estimated packet processing time for the packet scheduling. The proposed algorithm differs from the previous estimation-based pair queuing algorithm [9] that focuses on the fairness between independent flows and preserving the sequence of packets through the PFQ method. Experimental results show that the proposed scheduler increases the PE utilization when compared with the PFQ scheduler, while avoiding the need for many output buffers to sort packets of the same flow in the correct sequence, unlike the SNM scheduler.

Ⅱ. Proposed Packet Scheduler

Fig. 1 shows the architecture of the NP used to evaluate the proposed scheduler. This NP can process packets of the same flow in parallel using multiple PEs in the processor array (PA), and its overall operation is as follows. The packet scheduler (PS) receives information related to packet scheduling from the packet inspector (PI). The PS uses this information of the current packet and the estimated finish time of the packet of the same flow scheduled right before the current packet, in order to schedule the current packet so that its estimated finish time is larger than that of the packet right before it.

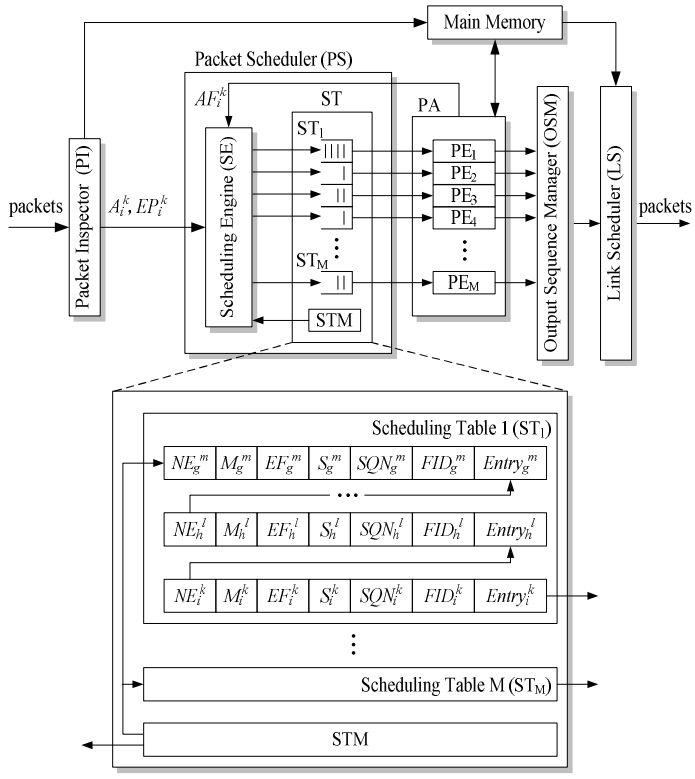

Fig. 1. Network processor architecture with the proposed scheduler

This scheduling is carried out based on the unit time (t) of the NP, which is the unit of below-mentioned times related to packet scheduling. If the actual finish time of packet processing differs from the estimated finish time, then the output sequence manager (OSM) sorts packets in the correct output sequence using output buffers.

For an incoming packet, the PI detects its flow id i and tags it with a unique sequence number k. Thereafter, the PI performs a deep packet inspection and estimates the packet processing time, then transfers the scheduling information of the packet, including the packet arrival time A_i^k, where the superscript k and subscript i denote the sequence number and flow id of the packet, respectively, and estimated packet processing time EP_i^k to the PS.

The scheduling engine (SE) in the PS calculates the start time of the packet processing S_i^k and estimated finish time of the packet processing EP_i^k. This process consists of three steps. First, to determine whether packet k of flow i should be scheduled with an additional time delay or not, the SE calculates the relative finish time RF_i^k, as shown in Eq. (1).

$$RF_i^k = A_i^k + \lambda + EP_i^k - EF_i^{k-1} \tag{1}$$

It is assumed that λ is a constant delay time between the packet arrival at the PI and the end of the packet scheduling in the PS. EF_i^{k-1} is the estimated processing time of the previous packet k-1 of flow i. Note that the main role of the SE is to schedule packets so that even though packets of the same flow are processed in parallel by different PEs, the correct sequence is maintained at the output of the PA. Thus, the SE adjusts S_i^k when RF_i^k is less than or equal to zero as shown in Fig. 2 using the Eq. (2).

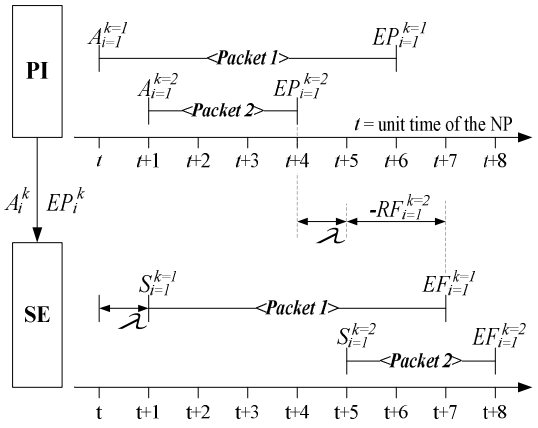

Fig. 2. Timing diagram of the packet scheduling

Finally, the SE calculates EF_i^k through the summation of Eq. (3) and keeps EF_i^k for use in the calculation of RF_i^{k+1}.

$$S_i^k = EF_i^{k-1} - EP_i^k + 1 \quad \text{if } (RF_i^k \le 0). \tag{2}$$

$$S_i^k = A_i^k + \lambda \quad \text{if } (RF_i^k > 0).$$

$$EF_i^k = S_i^k + EP_i^k \tag{3}$$

Each PE includes a scheduling table (ST) as a job queue for the packet processing. The SE stores the processing information of each packet, such as its start time S_i^k, estimated finish time EF_i^k, main memory address M_i^k, next entry address NE_i^k, flow id FID_i^k, and sequence number SQN_i^k, in the ST of its assigned PE.

The pseudo code in Fig. 3, which consists of two functions, describes the main operations of the proposed packet scheduler. The Scheduling_Engine function schedules packet k of flow i. The scheduling table manager (STM) points the last allocated entry in each ST, so that the SE can easily find the ST with the last entry that has the minimum estimated finish time of packet processing $EF_{\min}$. In addition, if $EF_{\min}$ is greater than or equal to the calculated S_i^k, S_i^k and EF_i^k are adjusted before the SE stores the processing information for packet k in the ST.

The actual finish time of packet processing AF_i^k can differ from EF_i^k, due to an incorrect EP_i^k or unexpected exception during the packet processing in the PE.

```
FUNCTION Scheduling_Engine
  Calculate S_i^k and EF_i^k using Eqs. (1), (2), and (3);
  FOR all STs
      Find the ST that has the minimum EF_min;
  IF EF_min ≥ S_i^k THEN
     Increase S_i^k and EF_i^k with the amount of (EF_min - S_i^k + 1);
  END
  Store the packet processing information in the next empty entry of the ST;
  END
ENDFUNCTION
FUNCTION Error_Adjustment
  FOR the ST to which packet k has been assigned
    IF AF ≥ S of the first entry of the ST THEN
      e = AF - S + 1;      n = first entry address of the ST;
      REPEAT
        Increase S and EF of the entry of address n with the amount of e;
        e = EF of the current entry - S of the next entry + 1; n = next entry address;
      UNTIL e < 0 or the last entry of the ST is reached
    END
  END
ENDFUNCTION
```

Fig. 3. Pseudo code of the proposed packet scheduler

If AF_i^k is less than EF_i^k, packet k is buffered in the OSM until the processing of the previous packets of flow i is finished. Conversely, if AF_i^k is greater than EF_i^k, the ensuing packets of flow i are buffered in the OSM until the processing of packet k is finished. Furthermore, in this case, the processing information stored in the ST to which packet k has been assigned needs to be adjusted, and the SE carries out this adjustment using the Error_Adjustment function.

If AF_i^k is greater than or equal to the S of the first entry in the corresponding ST, the S and EF of the first entry in the ST are adjusted based on the difference between the AF_i^k and S of the first entry in the ST. Subsequent entries in the ST are also adjusted based on the difference between the EF of the current entry and the S of the next entry until this difference is less than zero or the last entry in the ST is reached.

III. Experimental Results and Analysis

To compare the proposed processing-time-estimation (PTE) packet scheduler with the conventional SNM, PFQ, and P-ORR packet schedulers, four NP models, each of which adopts one of these schedulers, was implemented using C. For fair comparisons, all the NP models consist of one SE and the same number of PEs, and the same traffic patterns were applied to them. Packets were generated with variable lengths, ranging from 50 to 1500 bytes, as most Ethernet LANs use a maximum transfer unit (MTU) of 1500 bytes. In addition, five traffic patterns were used with a varying P_{sf}, the probability that packets of the same flow enter the NP consecutively (i.e. the probability of burst-type traffic patterns). P_{sf} varies from 0 to 1 with a step of 0.25, thus making five different traffic patterns.

While packet processing time depends on various factors, packet length is the key factor of packet processing time [9]. Thus, in the experiments, it was assumed that the actual packet processing time AP_i^k and estimated packet processing time EP_i^k are proportional to the packet length. However, to evaluate the influence of incorrect EP_i^k on the performance of the PTE, AP_i^k differs from EP_i^k with an error value of 10% and error rate P_e in the PTE experiments.

Fig. 4 shows the average utilization rates of the PEs as a function of the number of PEs. The experimental results showed that when the number of PEs was increased, the utilization rates of the PEs with the proposed PTE is saturated around 80% and the conventional SNM remained constant, while the utilization rate of the PEs with the PFQ and P-ORR decreased as the number of PEs increased.

This was mainly because in cased of the PFQ and P-ORR, increasing the number of PEs also increasing number of idle PEs since the conventional PFQ processes packets of the same flow sequentially using the same PE and the idle period on PEs between adjacent scheduling rounds increases in the P-ORR.

Meanwhile, in the case of the conventional SNM, packets of the same flow are scheduled to multiple PEs, regardless of the packet sequence, thereby requiring a large hardware overhead to sort packets of the same flow in the correct output sequence, in contrast to the proposed PTE.

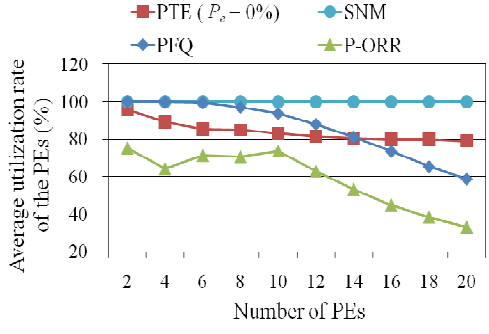

Fig. 4. Average utilization rate of PEs as function of the number of PEs

Fig. 5 shows the average number of packets per cycle in the output buffers of the OSM. Here, when the number of PEs is increased, the buffer size required by the conventional SNM and P-ORR rapidly increased. In contrast, when P_e ranged from 10% to 50%, the proposed PTE required a much smaller output buffer size than the conventional SNM and P-ORR.

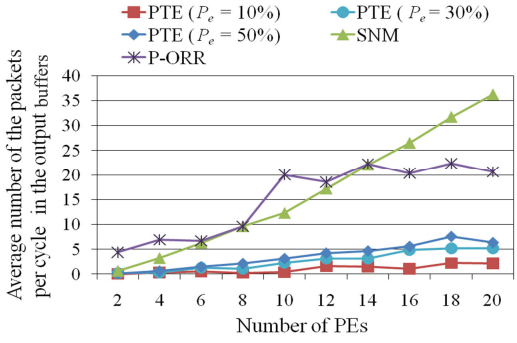

Fig. 5. Average number of the packets per cycle in the output buffers as a function of the number of PEs

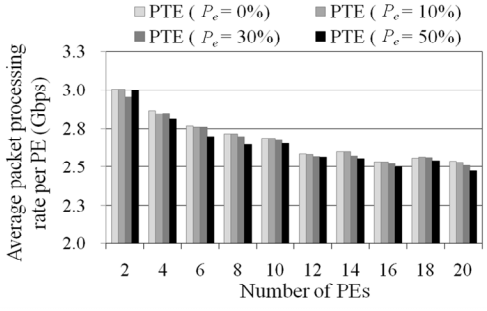

Fig. 6. Average packet processing rate per PE as a function of the number of PEs

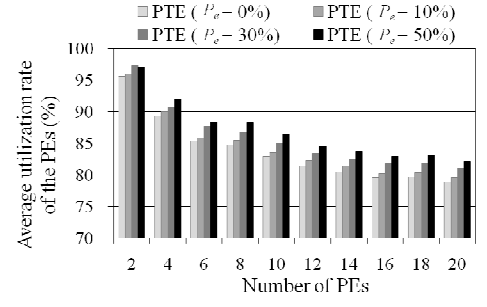

Fig. 7. Average utilization rate of the PEs as a function of the number of PEs

Figs. 6 and 7 show the influence of varying P_e on the performance of the proposed PTE. As shown in Fig. 6, varying P_e had a minimal influence on the average packet processing rate per PE for the following reasons. First, when AF_i^k was less than EF_i^k, this did not increase the packet processing rate, as there was no change in the scheduling information of the subsequent ST entries. Second, when AF_i^k was greater than EF_i^k, there was hardly any decrease in the packet processing rate, as the Error_Adjustment function in Fig. 3 generally offset the plus errors by filling any idle time slots between subsequent ST entries. Similarly, varying P_e had a minimal influence on the PE utilization rate. This is because as the P_e increased, idle time of packet processing between adjacent entries of the ST is decreased by the Error_Adjustment function of Fig. 3.

IV. Conclusion

This paper proposed a new sequence-preserving packet scheduler for an NP with multiple PEs. Using pre-estimated packet processing time and feedback from the actual packet processing time in the PEs, the proposed scheduler is able to preserve the correct output sequence of packets of the same flow, even though packets of the same flow are processed in parallel by different PEs. Experimental results show that the proposed scheduler improved the packet processing rate and PE utilization rate when compared with the conventional PFQ and P-ORR scheduler. As the number of PEs are increased to 20, the utilization rates of the proposed PTE is saturated to 80% while the utilization rate of PFQ and P-ORR decreased to 60% and 32% respectively. This performance improvement was also accomplished with only a small number of output buffers, unlike the conventional SNM and P-ORR scheduler. Furthermore, the proposed scheduler even performed well with high error rates of the packet processing time estimation. In future work,

an algorithm and hardware architecture will be developed for cost-efficient and accurate packet processing time estimation, which was not the concern of this paper.

References

[1] M. F. Iqbal, J. Holt, J. H. Ryoo, G. de Veciana, and L. K. John, "Dynamic Core Allocation and Packet Scheduling in Multicore Network Processors", in IEEE Transactions on Computers, Vol. 65, No. 12, pp. 3646-3660, Dec. 2016.

[2] A. Shpiner, I. Keslassy, and R. Cohen, "Scaling Multi-Core Network Processors without the Reordering Bottleneck", in IEEE Transactions on Parallel and Distributed Systems, Vol. 27, No. 3, pp. 900-912, 1 Mar. 2016.

[3] T. J. Kim and B. Suh, "A Packet Loss Differentiation and Guarantee in the Weighted fair Queuing", The Journal of Korean Institute of Information Technology, Vol. 11, No. 10, pp. 69-77, Oct. 2013.

[4] M. Laor and L. Gendel, "The effect of packet reordering in a backbone link on application throughput", Network, IEEE, Vol. 16, No. 5, pp. 28-36, Sep./Oct. 2002.

[5] Y. Qi, B. Xu, F. He, B. Yang, J. Yu, and J. Li, "Towards high-performance flow-level packet processing on multi-core network processors", Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems, Orlando, Florida, USA, pp. 17-26, Dec. 2007.

[6] N. Elnathan, "Reordering of out-of-order packets", United States Patent, US 7,072,342, Jul. 2006.

[7] S. Traboulsi, M. Meitinger, R. Ohlendorf, and A. Herkersdorf, "An efficient hardware architecture for packet re-sequencing in network processors mpsocs", 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, Patras, pp. 11-18, Aug. 2009.

[8] J. N. Yao, J. N. Guo, and L. N. Bhuyan, "Ordered round-robin: An efficient sequence preserving packet scheduler", IEEE T Comput, Vol. 57, No. 12, pp. 1690-1703, May 2008.

[9] T. Wolf, P. Pappu, and M. A. Franklin, "Predictive scheduling of network processors", Computer Networks, Vol. 41, No. 5, pp. 601-621, Apr. 2003.

Authors

Seung-Ho Ok

2006 : B.S., Mechatronics Engineering, Dong-eui Univerisy.

2008 : M.S., School of Electrical Engineering and Computer Science, Kyungpook National University.

2011 ~ 2013 : Visiting Scholar, Georgia Institute of Technology.

2014 : Ph.D., School of Electronics Engineering, Kyungpook National University.

2014 ~ 2017 : Senior Engineer, Samsung Electronics.

2017 ~ present : Assistant Professor, Dong-eui Univerisy.

Research interests : robot vision, SoC, VLSI.

Byungin Moon

1995 : B.S., Electronic Engineering, Yonsei University.

1997 : M.S., Electronic Engineering, Yonsei University.

2002 : Ph.D., Electrical & Electronic Engineering, Yonsei University.

2002 ~ 2004 : Research Engineer, SK Hynix.

2004 ~ 2005 : Research Professor, Yonsei University.

2005 ~ present : Professor, Kyungpook National University.

Research interests : SoC, computer architecture, vision processor.