An Efficient Packet Scheduling Algorithm in Network Processors

Jiani Guo, Jingnan Yao and Laxmi Bhuyan
Department of Computer Science and Engineering
University of California, Riverside
{jiani,jyao,bhuyan}@cs.ucr.edu

Abstract— **Several companies have introduced powerful network processors (NPs) that can be placed in routers to execute various tasks in the network. These tasks can range from IP level table lookup algorithm to application level multimedia transcoding applications. An NP consists of a number of on-chip processors to carry out packet level parallel processing operations. Ensuring good load balancing among the processors increases throughput. However, such multiprocessing also gives rise to increased out-of-order departure of processed packets. In this paper, we first propose a Dynamic Batch Co-Scheduling (DBCS) scheme to schedule packets in a heterogeneous network processor assuming that the workload is perfectly divisible. The processed loads from the processors are ordered perfectly. We analyze the throughput and derive expressions for the batch size, scheduling time and maximum number of schedulable processors. To effectively schedule variable length packets in an NP, we propose a Packetized Dynamic Batch-CoScheduling (P-DBCS) scheme by applying a combination of deficit round robin (DRR) and surplus round robin (SRR) schemes. We extend the algorithm to handle multiple flows based on a fair scheduling of flows depending on their reservations. Extensive sensitivity results are provided through analysis and simulation to show that the proposed algorithms satisfy both the load balancing and in-order requirements in packet processing.**

I. INTRODUCTION

With the advent of powerful network processors (NPs) in the market, many computation-intensive tasks such as routing table look-up, classification, IPSec, and multimedia transcoding can now be accomplished more easily in a router. Such an NP-based router permits sophisticated computations within the network by allowing their users to inject customized programs into the nodes of the network [1]. An NP provides the speed of an ASIC and at the same time is programmable. Each NP consists of a number of on-chip processors that can provide high throughput for network packet processing and application level tasks [2], [3], [4]. However, processing of packets belonging to the same flow by different processors gives rise to out-of-order departure of the packets from the NP and incurs high delay jitter for the outgoing traffic. For TCP, it has been proved that out-of-order transmission of packets is inimical to the end-to-end performance. For many applications like multimedia transcoding [5], it is imperative to minimize this out-of-order effect because the receiver may not be able to reorder them easily to tolerate high delay jitter.

This research has been supported by NSF grants CCF-0220096, CCF-0311437 and research grants from UC Micro and Intel Corporation.

Today's receivers vary widely from palm devices, PDAs to desktops that may or may not have enough storage and reordering capabilities. Examples of multimedia transcoding in an active router are found in the MeGa project [6] of the University of California, Berkeley, and the Journey network model [7] at the NEC-USA, where routers provide customizable services according to packet requests. Efficient packet scheduling is necessary in order to guarantee both high throughput and minimal out-of-order departures of packets. However, these two goals are contradictory to each other because scheduling on more number of processors increases throughput but also increases out-of-order departure of packets.

Packet processing in an NP can be considered as similar to link aggregation techniques that employ multiple physical links from a source to the same destination. Link aggregation provides increased bandwidth and reliability between the two devices (switch-to-switch or switch-to-station) as more channels are added. Implementations include the Cisco Etherchannel in the CISCO ONS 1500 Series based on the proprietary Inter-Switch trunking (ISL), Adaptec's Duralink port aggregation, 3COM, Bay Networks, Extreme Networks, Hewlett Packard and Sun, etc. A practical link stripping protocol, called Surplus Round Robin (SRR), is proposed by Adisesha [8] to schedule variable length packets over multiple links with different capacities. They demonstrate that stripping is equivalent to the classic load-balancing problem over multiple channels. They solve the variable packet size problem by transforming a class of fair queuing algorithms into load sharing algorithms at the sender. Although their solution is elegant and efficient, it requires the receiver to run a corresponding resequencing algorithm to ensure in-order delivery of packets.

The aim of this paper is to derive an efficient packet-scheduling algorithm in a network processor that comprises of a number of processors (or channels) for packet processing. It should provide 1) load balancing for processing variable length packets using a group of heterogeneous processors, and 2) in-order delivery of packets without considering receiver's rearranging capability. A plethora of scheduling schemes for multiprocessors have been proposed in parallel processing community. Examples of such schemes vary from simple static policies, such as round robin or random distribution policy [1], [9], [10] to sophisticated adaptive load-sharing policies [11], [5], [12]. However, only simple policies such

0-7803-8968-9/05/$20.00 (C)2005 IEEE

as round robin are employed in practice because adaptive schemes are difficult to implement and involve considerable overhead. We have shown that round robin is simple and fast, but provides no guarantee to the playback quality of output streams of video signals because it causes large out-of-order departure of the processed media units [5]. Adaptive load sharing scheme, which we implemented from the literature [11], achieves better unit order in output streams, but involves higher overhead to map the media unit to an appropriate node. Recently, we proposed a scheduling algorithm called Static Sequentialized Batch CoScheduling for video transcoding [13] using homogeneous network processors. It considered a single multimedia stream with all the packets available in the input queue at the beginning of the scheduling.

In this paper we derive a Dynamic Batch Co-Scheduling (DBCS) algorithm that considers a backlogged queue, and can be applied to dynamic arrival of packets in an overload situation. Expressions for load distribution in heterogeneous network processors are derived first by assuming that the schedulable workload is perfectly divisible in terms of bytes. The divisible load theory (DLT) for parallel processing has been suggested in [14], [15]. However, they did not consider sequential ordering of the processor execution times, as required for packet transmission over the output link. Our algorithm schedules the packets in batches by computing the optimal batch size, scheduling time, and number of schedulable processors given the maximum packet size and network processor parameters. A batch is similar to the concept of time epochs when scheduling is done. Several interesting results are derived regarding scalability of our algorithm.

Because the arriving packets cannot be distributed to processors in bytes, we also derive a packetized version of the DBCS algorithm by applying a combined version of DRR and SRR algorithms [16], [8]. The P-DBCS algorithm produces better results in terms of throughput and out-of-order rate compared to round robin and pure SRR schemes. We then extend the P-DBCS algorithm to handle multiple flows having reservations. When applying P-DBCS algorithm to multiple flows, both load balancing and fair scheduling requirements should be satisfied. Hence, we revise the expression of the batch size and packet dispatching condition to reflect the new requirements. Finally, we perform a number of simulations and sensitivity studies to verify the accuracies of our theory and obtain performance over wide-ranging input parameters.

The rest of the paper is organized as follows. In section II, we present the preliminaries and certain design issues in a Network Processor. In section III, we design the Dynamic Batch CoScheduling algorithm (DBCS) by doing theoretical derivation and analysis. In section IV, we propose and design a packetized version of the DBCS algorithm called Packetized-DBCS (P-DBCS) to deal with variable length packets. In section V, we present how to achieve fairness among multiple network flows using P-DBCS. Simulation results are presented in section VI in comparison with several other schemes. Finally, in section VII, we conclude the paper with future possible extensions related to this paper.

II. Model for Packet Scheduling

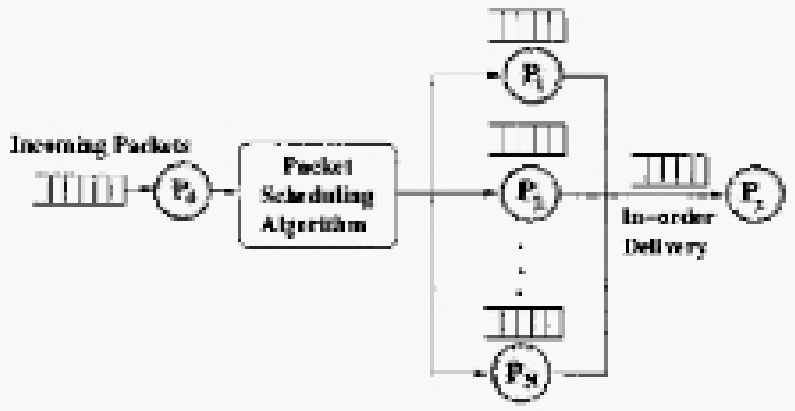

Fig. 1. Packet Scheduling in Network Processors

Figure 1 illustrates the multiprocessor architecture model of a router using a network processor (NP). The NP consists of one dispatching processor P_d, a few worker processors, P_1 through P_N, and a transmitting processor P_x. Intel IXP NP divides its set of microengines this way for packet processing [2]. The dispatching and transmitting processors communicate with the I/O ports sequentially. The dispatching processor P_d schedules incoming packets among the worker processors for packet processing. The transmitting processor P_x receives packets from processors P_1 through P_N and sends them to the output port. The aim of the packet scheduling algorithm is two fold: 1) the input load is balanced among the processors P_1 through P_N; 2) the flow order is maintained when the packets are transmitted to the transmitting processor P_x.

A similar problem called channel stripping has been addressed in the literature by Adiseshu [8]. There are N channels between the sender and the receiver. The sender implements the stripping algorithm SRR to strip incoming traffic across the N channels, and the receiver implements a resequencing algorithm to combine the traffic into a single stream. The stripping algorithm aims to provide load sharing among multiple channels. It does not consider the transmission order among the packets in different channels. Hence, the receiver needs to run a resequencing algorithm to restore the packet order in the original flow. A strict synchronization between the sender and the receiver is difficult to implement.

Although the packet scheduling problem looks different from the channel stripping problem, there are many similarities. First, in channel stripping, the packets are transmitted in the channels. In NPs, the packets are processed on the worker processors. These two times are equivalent and proportional to the load size in bytes. Second, in channel stripping, the time to move packets from the single input port to different channels is assumed to be negligible in [8]. Actually, there should be a time overhead in executing SRR at the IP level processing. As for packet scheduling in an NP, the time to move packets from the dispatching processor to a worker processor cannot be ignored because of the time taken by the dispatching processor and the transmission between the two processors. Finally, the transmitting processor in an NP removes the packets from the worker processors on an FCFS basis, whereas the receiving processor in the stripping model executes a

resequencing algorithm. Hence the models developed in this paper are applicable both to packet processing in an NP or packet transmission over multiple channels.

For the rest of the paper, we will explain our models/algorithms just in terms of NPs without loss of generality. To describe the whole procedure experienced by one packet when it is processed in an NP, there are three steps: 1) D-step: the dispatching processor dispatches the packet to a worker processor; 2) P-step: the worker processor processes the packet; 3) T-step: the worker processor sends the packet to the transmitting processor. Correspondingly, in the channel stripping problem, only one step, namely P-step, is modeled, if we take transmission of a packet as a type of processing. Hence, the packet scheduling problem in an NP is more complicated.

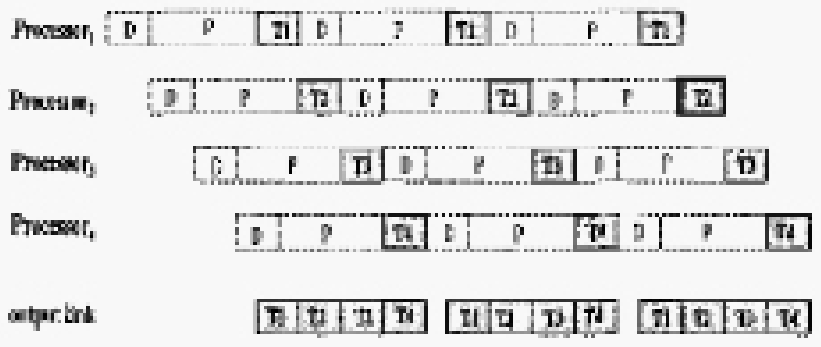

Fig. 2. Dynamic Batch CoScheduling

We propose a sequential completion pattern of the packet processing and thus a sequential data delivery by the worker processors, as illustrated in Figure 2. Let there be N worker processors in the router, each worker processor P_i $\forall i$, first receives some packets from the dispatching processor P_d (D-step), then processes these packets (P-step), and finally sends the packets to the transmitting processor P_x (T-step) sequentially. We propose to let the dispatching processor P_d distribute the packets among the N worker processors from P_1 through P_N in such a way that each worker processor completes processing and transmission sequentially. In another word, the worker processor P_i for (i=2 $\sim$ N) starts delivering the packets to the P_x immediately after P_{i-1} completes its T-step. Therefore, sequential packet delivery is ensured, as well as the load is balanced among multiple processors. We call a scheduling algorithm that produces such a scheduling pattern Dynamic Batch CoScheduling (DBCS). A nice property of the above model is that sequential data delivery is achieved without any additional control. Simply by arranging the computation and communication phases, a scheduling algorithm is obtained.

However, to design a practical algorithm, there are many issues to be addressed. First, how many packets should be dispatched to each worker processor to produce the desired pattern? Second, what if the packets are of variable lengths? Thirdly, given a network processor configuration and any packet arrival rate, can we always find a way to schedule packets in such a sequential delivery pattern? Fourthly, how to ensure fair scheduling among multiple flows having different reservations? Rest of the paper attempts to analyze the situations and provide answers to the above questions.

Table I defines the notations that are used throughout the rest of the paper. First, we devise a DBCS algorithm assuming that the workload is perfectly divisible. In the theoretical approach, a load distribution $\alpha = \{\alpha_1,...,\alpha_i, ..., \alpha_N)$ is first determined to produce the sequential scheduling pattern in one scheduling round. To further enable dynamic scheduling, we derive expressions for the *minimal batch size* I and the *Batch Size* B based on the maximal possible packet length L. Then, the complete DBCS algorithm is designed to schedule packets over multiple scheduling rounds with a care to always keep the sequential delivery of multiple batches of data. The scalability of the DBCS algorithm is analyzed and important conclusion reaches. To schedule variable length packets, a packetized version of the DBCS algorithm (P-DBCS) is devised based on the combination of DRR and SRR. In this way, we always minimize the absolute difference between the actual load distribution and the ideal load distribution. Finally, the P-DBCS algorithm is extended to handle multiple flows having reservations.

III. Dynamic Batch CoScheduling

In this section, we build a theoretical model for the general packet scheduling problem in an NP, and develop a load scheduling algorithm Dynamic Batch CoScheduling (DBCS) to produce a scheduling pattern described in Figure 2. There are two design goals of DBCS. The first is to ensure load balancing for a group of heterogeneous processors. The second is to ensure strict in-order delivery of data. In the following derivation, we assume that the input load is divisible at the granularity of one byte.

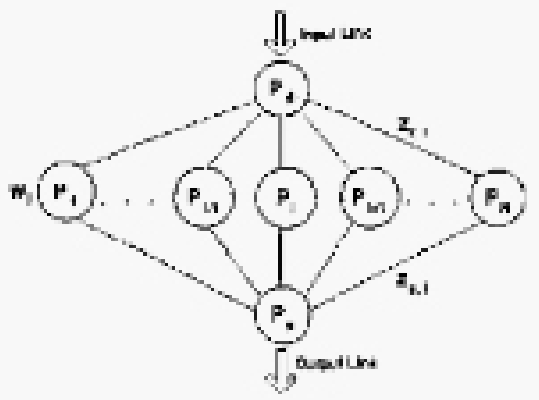

Fig. 3. Load Scheduling Model

A. Load Distribution in A Single Batch

For an NP, we set up the following mathematical model for the case of theoretical analysis. As shown in Figure 3, there are $(N + 2)$ processors and $2N$ links inside the router. The worker processors P_1, P_2,...,P_N are connected to the dispatching processor P_d via links $l_{r,1}$, $l_{r,2}$,...,$l_{r,N}$. Meanwhile, each worker processor has a direct link $l_{s,i}$ to the transmitting processor P_x. The dispatching processor receives packets from the input link, divides the input load into N parts and then distributes these load fractions to the corresponding worker processor. Each worker processor P_i starts processing immediately upon receiving its load fraction α_i and continues to do so until this fraction is finished. Finally, P_i sends the