A Formal Model for Network Processor Workload*

Xiao Ming Zhang, Zhi Gang Sun, and Min Xuan Zhang

School of Computer, National University of Defense Technology,
410073, Changsha, China
xiaomingzone@gmail.com

Abstract. Due to the heterogeneity of network processor architectures and constantly evolving network applications, it is currently a challenge to characterize the network processor workloads. In this paper, we formally model the task-level workloads of network processors as reactive dataflow process network (RDPN). RDPN is a suitable model of computation for formally describing the behaviors of packet-level parallel processing and event interaction with control point of network processors. We extend the expressive capability of RDPN by using three transformations (i.e., clustering, decomposing and duplicating) to analyze the model and support the further design space exploration of network processors.

1 Introduction

A router in core/edge networks is functionally split into control plane (also known as control point) and data plane. Network processors (NPs) are usually located on data-plane of routers and implements functional processing tasks, i.e. packet forwarding, security control and QoS (Quality of Service). Control plane and data plane of routers are interacted with each other in event-driven mode. A special case of interactive transaction would be that control plane constantly updates the forwarding tables of the data plane. These packet processing tasks and interactive transactions are called workloads of NPs. Due to the heterogeneity of network processor architectures and constantly evolving network applications, it is currently a challenge to characterize the network application workload.

Applications based on dataflow processing can be formally modeled by some MoC (Model of Computation) [1, 2, 3, 4, 5, 6]. In this paper, we introduce a novel dataflow process network (DPN) [7] model called Reactive DPN (RDPN) to model application workloads in NP domains. In RDPN model, reactive interfaces are added to basic DPN process and used for describing interactive events between control and data planes, while packet dataflow processing in data plane is still modeled as basic DPN. The remarkable work on this model is that we introduce three transformations (i.e., clustering, decomposing and duplicating) to analyze the model and support the further development of network processor design space.

* The work has been co-supported by National Sciences Foundations of China (NSFC) under grant No.2003CB314802, Hi-Tech Research and Development Program of China (863) under grant No.2003AA115130.

H. Jin, D. Reed, and W. Jiang (Eds.): NPC 2005, LNCS 3779, pp. 274 – 277, 2005.

The remainder of this paper is structured as follows. Section 2 presents the basic structure of our RDPN model. Section 3 then describes three transformations of RDPN. Section 4 explores some implementations of RDPN. Finally, we draw some conclusions in Section 5.

2 RDPN Model

The NP application workload represents a serial of packet processing functions. It can be naturally represented by DPN model, which makes parallelism and communication within an application explicit. A single functional task in the workloads is defined as a process and communicates with other processes through unbounded FIFO channels. All elements stored on the channels will be abstracted as tokens, including packets, events and control information.

However, DPN model has vulnerability to describing interactive behaviors between data plane and control plane because of its dataflow-aware feature. A DPN process referring to a task of data plane interacts with control plane through control channels, and with other process of data plane through data channels. We call this special DPN model as *reactive* DPN (RDPN). A single process model is illustrated in Fig. 1(a), where CI, CO respectively denote the input and output control channels between data and control planes, and DI, DO respectively denote input and output dataflow channels in data plane. The process consists of read channel actions, write channel actions and a set of internal function actions $\{ f_0, f_1, f_2 \cdots \}$ mapping input channels to output ones.

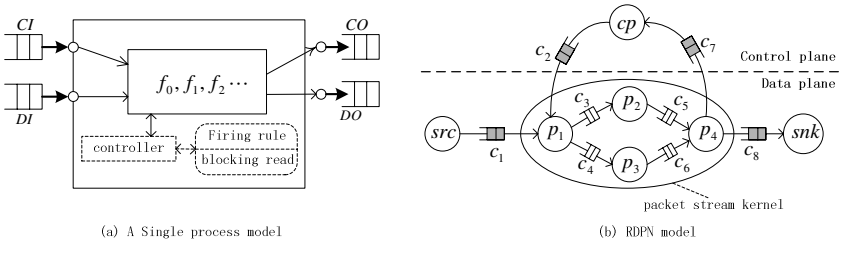

Fig. 1. The structure of our RDPN model, where (a) is the structure of a single process and (b) is the whole RDPN model of the packet processing structure in NP domains

Within our single process model, there is an implicit controller with the function firing rules and blocking read restriction drawn with dashed boxes. The controller itself checks firing rules of every function in a sequential order with blocking reads. When a function is activated, it reads input channels with blocking read and write output channels with non-blocking write. When all input arguments are present the function will be evaluated instantaneously. After evaluating the function, the controller checks the firing rules again until a valid firing is found.

From the single process model, the NP workloads can be described by the RDPN model, which is defined as multiple processes connected with each other according to

the process network topology. Our RDPN is an open system, which interacts with external environments through channels. Fig. 1(b) shows an example of RDPN for NP workloads, where *src* and *snk* processes respectively denote input and output packet streams of data plane, while *cp* process denotes the control plane. The packet stream kernel represents the packet processing workload, consisting of process $p_1 \sim p_4$ and channel $c_3 \sim c_6$. Channel c_1, c_2, c_7 and c_8 are environmental interactive channels.

When there is no interaction between data and control planes, the behaviors of our RDPN model act as DPN. There are two types of interactive events between data and control plane: *down events* are used for control plane downloading control information (i.e. NP configuration or routing update) to data plane; *up events* are used for data plane uploading local information (i.e. local states or packets) to control plane. Down events will disrupt the pipelining of packet flow and deadlines of packet processing, while up events not do so. We handle these two types of interactive events in two different ways: (1) the up events are only treated in the same way as normal dataflow throughout the RDPN. (2) we use *event reactive point* (ERP) to control the moment when down events would be applied within the data plane. In more details, before processing an input event the packet input to the network conceptually needs to be frozen and all data must be processed internally. Only when all data has been processed, the event can be applied and the dataflow can be continued.

3 Transformation of RDPN

RDPN model is constructional and hierarchical. We address three transformations of RDPN: clustering, decomposing and duplicating. Clustering and decomposing are used to support hierarchical modeling. Duplicating is used to develop task-level parallelism on the network processor architectures.

When multiple processes communicate with each other frequently or exchange a large amount of dataflow between them, clustering these processes into a single process can avoid the overhead of communication through channels.

A complex process in RDPN can be discomposed into several relatively simple processes in order to analyze the feature and structures of RDPN and develop the task-level parallelism of the network processor workload. We assume that a single process of RDPN is a program described with some kind of programming language (such as C /C ++). Thus decomposing of a process is the same action as translating a program into RDPN, which provides a bridge between the actual workload programs and our formal RDPN model.

A complex task would be distributed to multiple components for parallel implementation. Duplicating a single process of RDPN is used to support this situation. To transform a process into multiple duplicates, we introduce a *switch* process which plays the role of schedulers to map input channels to the duplicates, and a *select* process which selects one duplicate to output its processing results. A RDPN model with duplicating structure can be easy to be mapped into NP architectures based on multi-processors.

4 Implementation of RDPN

We implement RDPN software framework in C++ programming environment by using concepts of object-oriented approach. All Processes of RDPN are implemented by classes. The behaviors of the classes depend on the operational semantics of RDPN. Actions in RDPN are implemented by member functions of the classes. Similarly, all channels are implemented as classes in which these channels are implemented by dynamic lists with FIFO operation.

In our RDPN software framework, every process is treated as an instantiation of its corresponding class, which is assigned a separate thread of execution. To implement event interactions between data and control planes, three threads are introduced to monitor *src*, *snk* and *cp* process in NP domains (referring to Fig. 1(b)). In the RDPN framework, the run-time environment coordinates the execution of all these threads.

5 Conclusion

In this paper, we have proposed a formal model for NP workloads, called RDPN. Our RDPN is used to describe packet processing behavior in data plane as well as the interactive transactions between data and control planes. Transformations of RDPN can extend the expressive capability of RDPN and support design space exploration of NP architectures. Furthermore, modeling NP workloads as RDPN is a start-point of our design space exploration of NP architectures. The next work will be to use the model for application optimization, allocation of processing tasks, developing novel NP architectures and mapping applications to NP architecture.

References

1. G. Kahn. The Semantics of a Simple Language for Parallel Programming. In J.L. Rosenfeld, editor, Information Processing 74, Proceedings, pages 471–475, Stockholm, Sweden, August 1974. North-Holland, Amsterdam, The Netherlands, 1974
2. E. Lee. Overview of the Ptolemy project. Technical Memorandum UCB/ERL No. M01/11, University of California, EECS Dept., Berkeley, CA, March 2001
3. A. Girault, B. Lee, and E. Lee. Hierarchical finite state machines with multiple concurrency models. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 18(6):742-760, June 1999
4. K. Strehl, et al. FunState - an internal design representation for codesign. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 9(4):524–544, Aug. 2001
5. B. Kienhuis, E. F. Deprettere. Modeling Stream-Based Applications using the SBF model of computation. IEEE Workshop on Signal Processing Systems (SIPS 2001), Antwerp, Belgium, September 26-28, 2001
6. M.C.W. Geilen, T. Basten. Reactive Process Networks. In Fourth ACM International Conference on Embedded Software, Proceedings, pages 137–146. Pisa, Italy, 27-29 September, 2004. ACM Press, New York, NY, USA, 2004
7. E. A. Lee and T. M. Parks. Dataflow Process Networks. Proceedings of the IEEE, May 1995. (http://ptolemy.eecs.berkeley.edu/papers/processNets)