Internet as a Dataflow Computer

Hiroyuki Ohsaki[1], Hideaki Suzuki[2], and Hidefumi Sawai[2]

[1] Graduate School of Information Science and Technology, Osaka University, Japan
oosaki@ist.osaka-u.ac.jp
[2] Kobe Advanced ICT Research Center (KARC), National Institute of Information and Communications Technology (NICT), Japan
{suzuki,sawai}@nict.go.jp

Abstract. This paper proposes a novel dataflow architecture called *DFNET (DataFlow architecture on the interNET)*, which realizes a scalable dynamic dataflow computer on a packet switching network. A vast amount of research on dataflow computers has been extensively performed during, in particular, 1970s and 1980s as a promising approach for realizing very high-speed computers. In spite of the large amount of expectations, success of dataflow computer researches is quite limited. The objective of this paper is to present the concept of a scalable and extensible dataflow architecture on a packet switching network, which utilizes the abundant resources of a large-scale computer network. In this paper, we introduce the concept of DFNET (DataFlow architecture on the interNET). DFNET is composed of configuration and control methodologies of routers in a packet switching network. The key of DFNET is that a packet switching network is utilized not for *end-to-end communication* but for *dataflow computing*. Because of desirable properties of a packet switching network, DFNET has high scalability in terms of the dataflow program size and high robustness against failures.

1 Introduction

Dataflow computer is computer architecture, which is significantly different from the conventional von Neumann architecture. In dataflow computers, computing is driven not by the program counter but by data themselves [1,2]. Dataflow computer is expected to solve the performance bottleneck of the von Neumann architecture with massively parallel program execution. A vast amount of research on dataflow computers has been extensively performed during, in particular, 1970s and 1980s as a promising approach for realizing very high-speed computers [1,2]. In von Neumann computers, which are also referred as *control flow* systems, every instruction pointed by the program counter is sequentially fetched by a processor for execution. On the contrary, dataflow computers have no program counter. In dataflow computers, operations are *fired* immediately when all required data (i.e., operands) are available.

In what follows, a classical data-driven dataflow model proposed by Dennis *et al.* [3] is briefly explained. In dataflow computers, a program to be executed is represented as a directed graph (*dataflow program*), which is composed of nodes (i.e., data and functions) and arcs (i.e., data flows) (Fig. 1). Several tokens, each of which holds datum to be processed, are transferred on arcs connecting nodes. Data in those tokens are updated at every node. A node without input arcs generates a datum; i.e., a token containing the datum is created and it is then transferred to the downstream node. A node with one or more input arcs receives tokens from upstream nodes. Once all tokens are received, the node performs a predefined operation for data contained in tokens. A token containing the result is created and it is then transferred to the downstream node. There are two types of arcs: *data arc* and *control arc*. The data arc carries tokens with arbitrary datum whereas the control arc does token with a Boolean value.

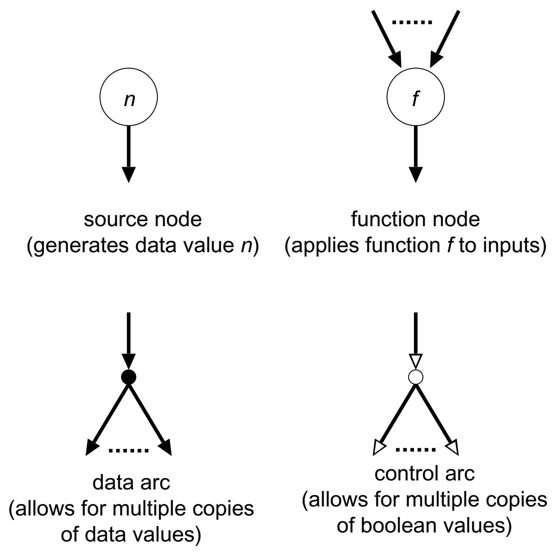


Fig. 1. Primitive nodes in Dennis's notation [3]

In 1970s and 1980s, the main concern of dataflow computer researches was primarily to build a much faster computer than the conventional von Neumann computers. A number of dataflow architectures and prototype systems have been published (see [1] and references therein). The key for realizing a very fast dataflow computer is in efficient parallel processing of program execution. In the literature, there exist several approaches for accelerating parallel processing of program execution. The classical dataflow architecture proposed by Dennis *et al.* is classified as *static dataflow architecture* or equivalently *single-token-per-arc* architecture [1,2]. In static dataflow architecture, only a single token can be transferred through an arc. Static dataflow architecture therefore limits the parallelism of program execution; i.e., loops and recursions must be performed sequentially. Inefficient parallelism of static dataflow architecture results in significantly slow program execution. *Dynamic dataflow architecture* extends static dataflow architecture to allow multiple tokens on an arc by, for example, adding a tag to every token or replicating a subgraph of the dataflow program [1,2]. Dynamic dataflow architecture therefore enables simultaneous execution of loops and recursions, which significantly accelerates the program execution. Several dynamic dataflow systems have been proposed and studied in the literature [1,2], in which either the tag format or the replication method of a dataflow subgraph is different.

In spite of the large amount of expectations, success of dataflow computer researches is quite limited. Dataflow computers were originally expected as a new computer architecture, which hopefully excelled the limitation of the conventional von Neumann architecture. The von Neumann architecture, however, has been continuously improved with several technological innovations such as the advancement and enhancement of semiconductor technologies. Researches on the von Neumann architecture have successfully caught up with the growing demand of computing resources. On the other hand, researches on dataflow computers have faced a difficulty. For instance, software development environment for dataflow programs has not been fully matured. Programming languages for dataflow programs are generally designed based on a *single static assignment paradigm*, which considerably limits the freedom of software development. Also parallel compilers for dataflow programs is not easy to develop, so that users of dataflow computers are little benefited from massively parallel program execution.

In this paper, we propose a novel dataflow architecture called *DFNET (DataFlow architecture on the interNET)*, which realizes a scalable dynamic dataflow computer on a packet switching network. Note that the objective of this paper is not to present yet another architecture for realizing a *high-speed* dataflow computer. Instead, this paper aims to present the concept of a *scalable and extensible* dataflow architecture on a very large-scale packet switching network. A packet switching networks such as the Internet is *best-effort*. Namely, QoS (Quality of Service) such as the speed and the delay of data transfer is not guaranteed. Conversely, packet switching network achieves efficient utilization of networking resources by scarifying the communication quality. It is therefore not desirable to build a dataflow computer on a packet switching network if one needs a very high-speed dataflow computer. We believe that a packet switching network such as the Internet, which has been exponentially expanding both in speed and size, should be a viable platform for realizing a (not so fast but) very large-scale dataflow computer.

DFNET is composed of configuration and control methodologies of routers in a packet switching network. A packet switching network has high scalability in terms of the network size (i.e., the number of routers and links) as well as high robustness against router and/or link failures. The key of DFNET is that a packet switching network is utilized not for *end-to-end communication* but for *dataflow computing*. Because of desirable properties of a packet switching network, DFNET has high scalability in terms of the dataflow program size and high robustness against failures. Also, utilization of the dynamic routing mechanism and network virtualization technologies realize a highly flexible and extensible dataflow computer.

Note that this paper only covers the concept and high-level view on building blocks of DFNET. In other words, several practical issues such as implementation, deployment, security, and software development environment are not covered. The core of DFNET assumptions is that the underlying network is a packet switching network. Although detailed discussion on DFNET implementation is beyond the scope of this paper, it should be worth considering possible DFNET implementations. DFNET might be implemented, for instance, either as an extension to IP routers [4], an application-level overlay network [5], or an application for active networking architecture [6].

The organization of this paper is as follows. Section 2 presents the overview of DFNET (DataFlow architecture on the interNET). We also explain four building blocks of DFNET: *token communication mechanism*, *token synchronization mechanism*, *data processing mechanism*, and *token routing mechanism*. Finally, Section 3 concludes this paper and discusses future works.

2 DFNET (DataFlow architecture on the interNET)

2.1 Overview

We first introduce the overview of DFNET (DataFlow architecture on the interNET), which is a novel architecture for realizing a scalable and dynamic dataflow computer on a packet switching network.

DFNET is essentially one of token-based dynamic dataflow architectures. DFNET builds a dataflow computer on a packet switching network, which is composed of several routers and links.

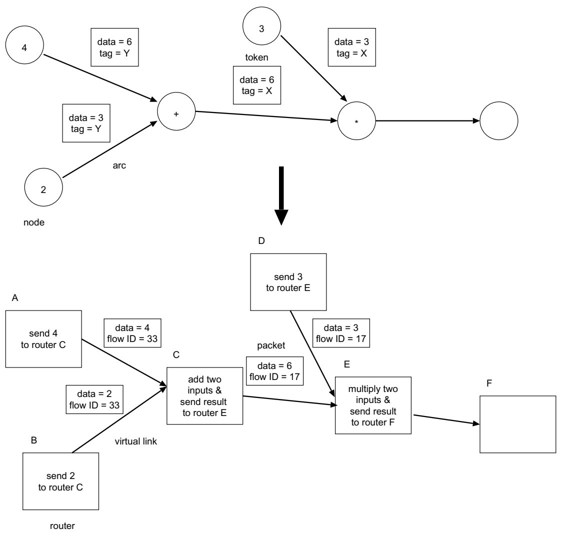


Fig. 2. DFNET (DataFlow architecture on the interNET) overview; in DFNET, a dataflow program is directly mapped to a packet switching network

In DFNET, a dataflow program is directly mapped to a packet switching network. Tokens, nodes, and arcs in a dataflow program are mapped to *packets*, *routers*, and *virtual links* in a packet switching network. A set of nodes in a dataflow program are assigned to a router in a packet switching network. A token in a dataflow program is encapsulated in a packet, and packets containing tokens are transferred between routers. The key of DFNET is that a packet switching network is utilized not for *end-to-end communication* but for *dataflow computing*.

In conventional dataflow architecture, such as MIT's Dynamic Architecture [7] and Manchester Architecture [8], separate PEs (Processing Elements) are in charge of data processing. Multiple PEs are therefore interconnected through a communication network. On the contrary, routers in DFNET performs both data processing and communication. In other words, PEs of a dataflow computer are embedded in the router. Integration of data processing and communication in a router greatly simplifies the architecture. It also enables direct mapping of a dataflow program onto a packet switching network. In DFNET, a dataflow program to be executed is split into multiple subgraphs, each of which is assigned to a router. In the followings, we will explain the case of one-to-one mapping of a node in the dataflow program to a router in the packet switching network. The case of many-to-one mapping can be easily realized by utilizing loopback interfaces in a router (i.e., by transferring a token within the router).

DFNET has four building blocks: *token communication mechanism*, *token synchronization mechanism*, *data processing mechanism*, and *token routing mechanism.*

2.2 Token Communication Mechanism

In DFNET, token communication from an upstream node to a downstream node in a dataflow program is realized by encapsulating a token in a packet and transferring the token between routers (see Fig. 3). Specifically, the address of a router, to which the downstream node is assigned, is written in the destination address field of the packet header. Also the token containing datum and tag is stored in the payload of the packet.

In a packet switching network, the destination address field of the packet header usually stores the address of a destination host for realizing end-to-end communication. On the contrary, DFNET stores the address of the next router in the destination address filed of the packet header. Namely, the packet switching network is utilized not for end-to-end communication between hosts but for hop-by-hop communication among routers.

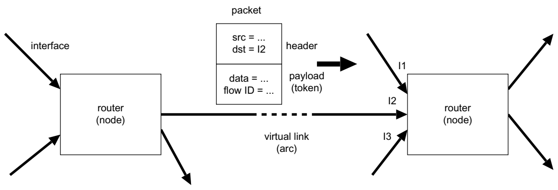


Fig. 3. DFNET token communication mechanism; token communication from an upstream node to a downstream node in a dataflow program is realized by encapsulating a token in a packet and transferring the token between routers

2.3 Token Synchronization Mechanism

In dataflow computers, a node is *fired* immediately when tokens from all input arcs are available. DFNET uses a *flow table* at a router for token synchronization (see Fig. 4); i.e., every token is assigned a globally unique identifier called *flow ID*, and a router maintains availability of tokens by updating records corresponding to their flow ID's.

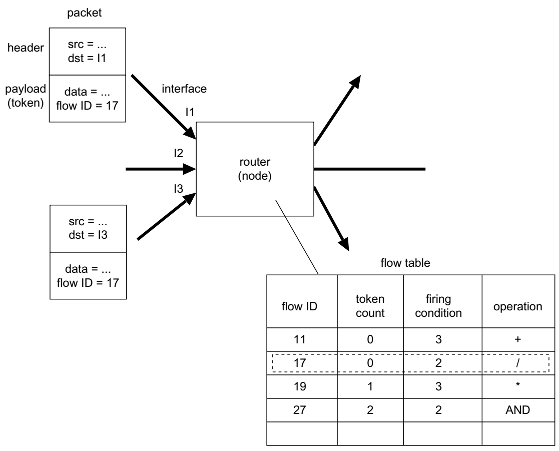
For token synchronization at a router, every token is assigned a unique flow ID. A record of the flow table corresponding to the flow ID represents *token count* (i.e., the number of tokens received) and *firing condition* (i.e., the number of token required for firing). The flow ID, token count, and firing condition are determined from the dataflow program.

When a router receives a packet, the router searches the flow table for the flow ID specified in the payload of the packet. The router then increments the token count of the record in the flow table. If the token count is equal to the firing condition of the record, the router is *fired*; i.e., the data processing mechanism is invoked and the token count is set to zero.

2.4 Data Processing Mechanism

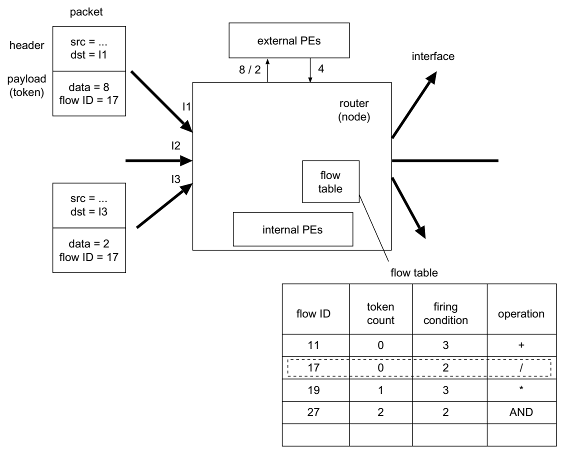
In DFNET, capability of PEs are embedded in a router; i.e., data processing is performed at the router. Data processing at the router can be realized either by utilizing internal computing resource in the router or providing external computing resource outside of the router (see Fig. 5).

In the case of internal computing resource, an operation for tokens is determined by the synchronization mechanism. One of internal PEs performs the specified operation, and the result is sent to the token routing mechanism, which will be explained below.



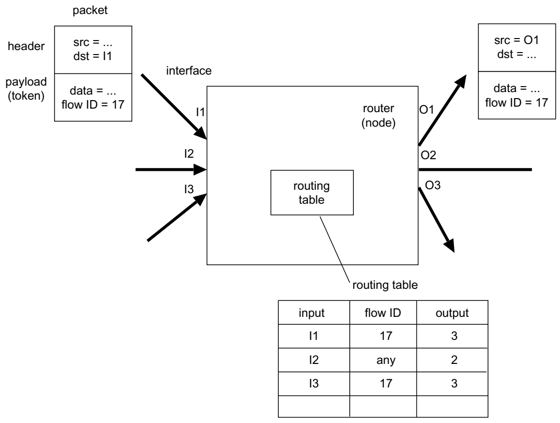
flow ID	token count	firing condition	operation
11	0	3	+
17	0	2	/
19	1	3	*
27	2	2	AND

Fig. 4. DFNET token synchronization mechanism; every token is assigned a globally unique identifier called *flow ID*, and a router maintains availability of tokens by updating records corresponding to their flow ID's



flow ID	token count	firing condition	operation
11	0	3	+
17	0	2	/
19	1	3	*
27	2	2	AND

Fig. 5. DFNET data processing mechanism; data processing at the router can be realized either by utilizing internal computing resource (internal PEs) or providing external computing resource (external PEs)



input	flow ID	output
I1	17	3
I2	any	2
I3	17	3

Fig. 6. DFNET token routing mechanism; DFNET utilizes a routing table in the router for determining the next-hop router, to which the downstream node in the dataflow program is assigned

Also in the case of external computing resource, an operation for tokens is determined by the synchronization mechanism. One of external PEs receives data and type of operations, and the result is sent to the token routing mechanism.

Note that both internal and external PEs can be used according to the type of data processing. For instance, for simple and/or realtime operations, internal PEs would be appropriate. On the other hand, for complex and non-realtime operations, external PEs would be appropriate. Introduction of external PEs slightly complicates the router architecture while the flexibility and expandability of the dataflow computer can be realized.

2.5 Token Routing Mechanism

DFNET utilizes a routing table in the router for determining the next-hop router, to which the downstream node in the dataflow program is assigned (see Fig. 6).

Each record of the routing table is a triplet of input interface, flow ID, and output interface. Based on the input interface of an arriving token and the flow ID written in the token, the router determines the output interface from the corresponding record of the routing table. If multiple output interfaces are found, the router replicates the token, and sends those tokens through all output interfaces.

Each record of the routing table is directly determined from the dataflow program. Note that change of the dataflow program during program execution can be easily realized by dynamically modifying the routing table.

3 Conclusion

In this paper, we have proposed a novel dataflow architecture called *DFNET (DataFlow architecture on the interNET)*, which realized a scalable dynamic dataflow computer on a packet switching network. The objective of this paper was not to present yet another architecture for realizing a *high-speed* dataflow computer. Instead, this paper aimed to present the concept of a *scalable and extensible* dataflow architecture on a very large-scale packet switching network. DFNET is composed of configuration and control methodologies for routers in a packet switching network. In this paper, we have explained four building blocks of DFNET: token communication mechanism, token synchronization mechanism, data processing mechanism, and token routing mechanism. Because of desirable properties of a packet switching network, DFNET has high scalability in terms of the dataflow program size and high robustness against failures.

Our future work includes architectural comparison of DFNET with other dynamic dataflow architectures, designing router architecture for DFNET, qualitative and quantitative performance analysis of DFNET, and experiments with prototype implementation of DFNET.

Moreover, extension of DFNET to support several types of different computing models than dynamic dataflow architecture would be of great importance. In [9,10,11], we have proposed ATN (Algorithmically Transitive Network), which is a self-organizing dynamic dataflow network with a learning mechanism. ATN adopts a BP (Back Propagation) learning mechanism similar to that of neural networks. DFNET has been initially designed as an execution environment for ATN. We should note, in particular, that such a BP-based learning mechanism can be easily implemented with the token routing mechanism in DFNET.

References

1. Sharp, J.A. (ed.): Dataflow computing: theory and practice. Ablex Publishing Corporation (1992)
2. Silč, J., Robič, B., Ungerer, T.: Processor architecture: from dataflow to superscalar and beyond. Springer, Heidelberg (1999)

3. Dennis, J.B., Misunas, D.P.: A preliminary architecture for a basic dataflow processor. In: Proceedings of the 2nd Annual Symposium on Computer Architecture, pp. 126–132 (1975)
4. Baker, F.: Requirements for IP version 4 routers. Request for Comments (RFC) 1812 (June 1995)
5. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys and Tutorials 7, 72–93 (2005)
6. Bhattacharjee, S., Calvert, K.L., Zegura, E.W.: An architecture for active networking. Tech. Rep., College of Computing, Georgia Institute of Technology, Atlanta, Georgia (1996)
7. Gostelow, A.K.P., Plouffle, W.: An asynchronous programming language and computing machine. Tech. Rep. 114a, Department of Information and Computer Science, University of California, Irvine, CA (1978)
8. Gurd, J.R., Watson, I.: A multilayered dataflow computer architecture. In: Proceedings of 7th International Conference on Parallel Processing (August 1977)
9. Suzuki, H., Ohsaki, H., Sawai, H.: A Network-Based Computational Model with Learning. In: Calude, C.S., Hagiya, M., Morita, K., Rozenberg, G., Timmis, J. (eds.) UC 2010. LNCS, vol. 6079, pp. 193–193. Springer, Heidelberg (2010)
10. Suzuki, H., Ohsaki, H., Sawai, H.: Algorithmically Transitive Network: a new computing model that combines artificial chemistry and information-communication engineering. In: Proceedings of the 24th Annual Conference of Japanese Society for Artificial Intelligence (JSAI), pp. 2H1–OS4–5 (2010)
11. Suzuki, H., Ohsaki, H., Sawai, H.: An agent-based neural computational model with learning. In: Conference Abstract of the 3rd INCF Congress of Neuroinformatics (Neuroinformatics 2010) (2010), doi:10.3389/conf.fnins.2010.13.00021