English translation of the paper:
V. V. Vlassov, A.V. Kraynikov, and B.A. Kurdikov, A Data Flow Computer System. In: Izvestiya LETI (Proceedings of Leningrad Electrotechn.Inst.), St.Petersburg, Vol. 436, 1991, pp. 3-7. (in Russian).

UDK 681.325.023:681.3.019

V.V.VLASOV, A.V.KRAYNIKOV, B.A.KURDIKOV

A DATAFLOW COMPUTER SYSTEM

Dataflow computing is a subject of considerable interest, since this class of computer architectures exposes high level of parallelism.

A dataflow computer system VSPD-1 presented in this paper is a research prototype of a static dataflow architecture according to the Veen's classification[1]. Static partitioning, linking, and distribution of actors among processing modules must be done at the stage of development of a dataflow application for VSPD-1.

VSPD-1 is designed to scale up to 16 processing nodes. The research prototype of VSPD-1 implemented at the Computer Engineering Department, LETI (Leningrad Institute of Electrical Engineering), consists of five processing modules (PM).

Each VSPD-1 processing module consists of

- a microcomputer MS11200.5 with a 1MIPS microprocessor and 32-KB RAM;
- PROM for a loader and a kernel of a dataflow run-time system (called here monitor);
- communication Qbus-Qbus adapters (CA);
- a I/O processor (IOP), that supports up to eight DMA channels for internode communication
- an auxiliary control unit that implements most of operations on dataflow control[2].

Restrictions of physical realization motivated the following configuration of the 5-modules system: four processing modules are configured in a ring, and the fifth module is connected with all other PMs. The fifth PM uses a communication adapter that connects the processing module to a host computer DVK-4. The structure of the VSPD processing module is presented in Fig.1.

Dataflow computation is realized at the level of labeled code fragments called *actors*. An actor is ready and can be performed, when it receives all data (operands) required for its execution. An application for VSPD-1 must be written in a specific dataflow style where actors are programmed in Pascal with macros in Macro-11 assembler. Each labeled code segment that specifies an actor ends with an assembler macro which passes a pointer to a destination list to the kernel and invokes it. The kernel sends data produced by the actor to destination actors specified in the list.

A label of the actor serves as its starting address (activation point). Addresses of destination actors that will receive the result tokens are stored in a destination table that actually holds a dataflow graph of the application. A dataflow program includes also a table of starting addresses of actors, and a table of ready bit vectors. A ready-bit vector indicates for each of actor inputs whether a token has arrived at that input. Before computation starts, the ready bit table contains initial mapping of tokens to actor inputs. A special mask stored together with a ready-bit vector indicates the inputs which are always ready (constant or not in use).

---

1. A.H.Veen, Dataflow machine architecture, ACM Computing Surveys, Vol. 18, No. 4, 1986, pp. 365-396.
2. As the control unit (called TokenFlow Unit) was developed as an optional module, functionality of this unit was also implemented in software as a part of the VSPD-1 run time system. - *Vlad Vlassov*

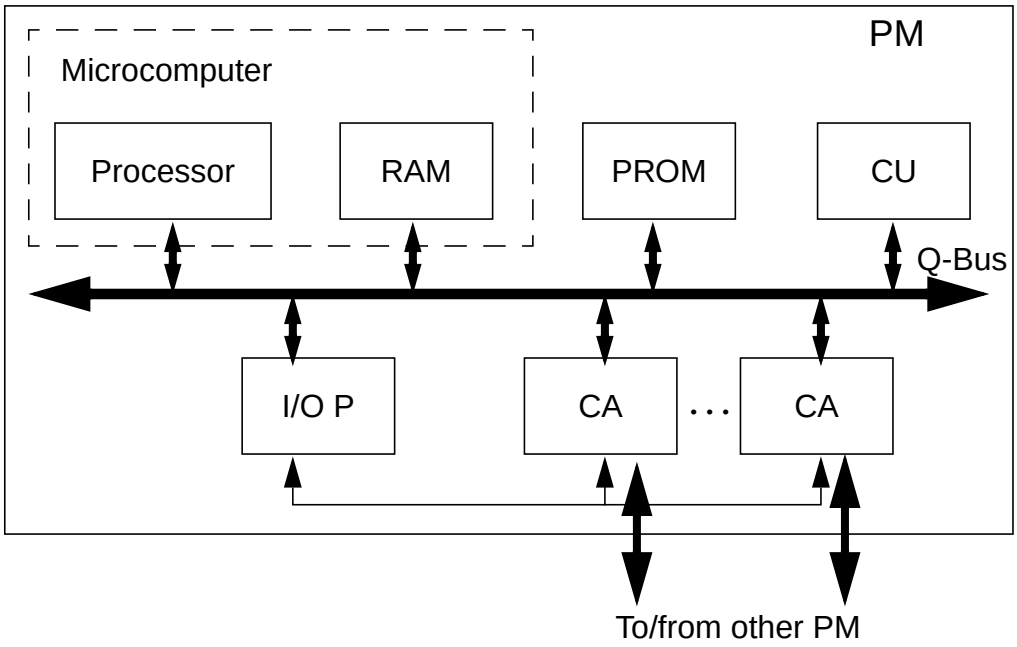

Fig.1.

The system supports explicit and implicit transferring data from source-actors to destination actors. With explicit transfer, data that are sent to a destination actor, are followed by a result token, whose role is to indicate that data has been sent to the destination. A token is 16-bits wide and consists of the following fields: the data type tag (T) that indicates whether the data item sent is a vector or a scalar; the number of the destination processor module (P); the number of the destination actor (A); the number of the actor input to which the token is directed (I). In this way, parameters for explicit data transfer consist of two words: a 16-bits token and a pointer to a data structure to be sent to the destination. The format of a destination list entry is shows in Fig.2.

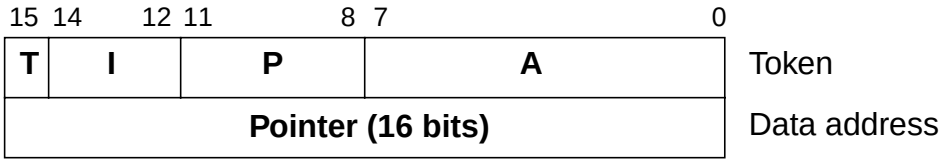

Fig.2.

Implicit data transfer is used for local data flow between actors within a processing module, and it is implemented through shared variables.

Tokens are used for synchronization of parallel processes in processing modules. Tokens that follow data, form a control flow since each token indicates which destination actor can be performed after a source actor. The actor is ready when it has collected tokens on all its inputs. The amount of token traffic can be less than that of data flow. For example, when a source actor sends more than one data item to a destination actor, it is enough to send only one token after all data are sent in the vector form. If data is directed to remote actors located on the same remote node, data and all tokens are sent in one message.

Token flow and data flow are controlled by a distributed software run-time system together with the auxiliary hardware Control Unit (CU) in each processing node. The auxiliary Control Unit consists of a register file (RF), a table memory (TM), a operational unit (OU), a ready-actor queue (RAQ), microprogrammed controller (MPC), address selector (AS) and bus adapters (BA). The table memory is split into three regions that are used to store three tables: the table of starting addresses; the table of ready-bit vectors, and the table of destination addresses. A structure of CU is shown in Fig.3.

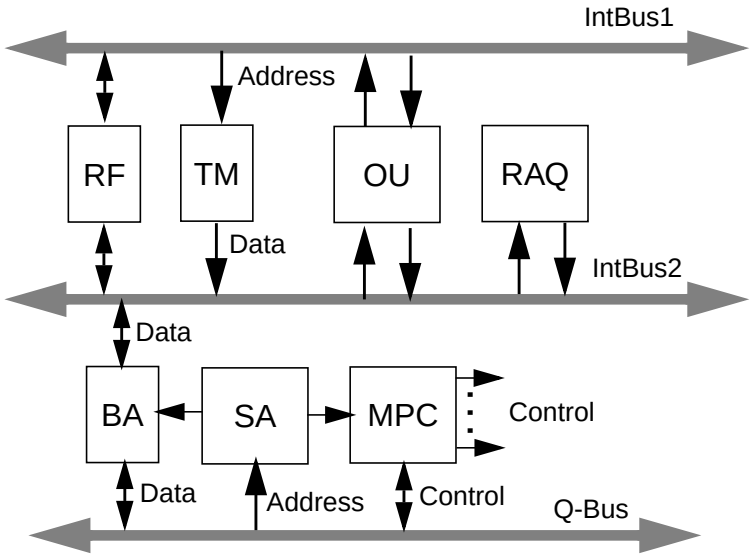

Fig.3.

The auxiliary Control Unit receives a token and sets an appropriate ready-bit in the read-bit table to indicate that the token has arrived. If the destination actor is ready, its number is inserted to the ready actor queue. On a request from the software kernel, CU fetches a ready actor (if any) from the head of the queue, and looks up the starting address table. A starting address of the ready actor is returned to the kernel that invokes the actor. After the actor completes, the kernel resets actor's ready-bit vector masked by a always-ready mask, and sends results and tokens to destination actors according to a destination list stored in the destination address table of CU.

Sending of data and tokens to remote processing modules is performed by a I/O co-processor[1] and communication adapters. A message that includes data (vector or scalar), is passed to the destination processing unit via DMA. When data transmission completes, a corresponding token is passed to the auxiliary Control Unit of the destination processing module. A DMA engine is controlled by the IO co-processor.

The VSPD-1 system can be used, in particular, as a functional accelerator for a minicomputer with a PDP-11 architecture for applications that require high performance computing, such as simulation of complex dynamic objects in real-time.

1. The I/O co-processor was developed as an optional module, its functionally was also implemented in software as a part of the VSPD-1 run-time system. - *Vlad Vlassov*