Procedure Level Data Flow Processing on Dynamic Structure Multimicroprocessors

TATSUO SUZUKI,* KEN KURIHARA,*
HIDEHIKO TANAKA* and TOHRU MOTO-OKA*

TOPSTAR, a highly parallel multi-microprocessor system is designed and constructed. It executes procedure level data flow processing (one data flow node is one procedure). TOPSTAR is composed of two kinds of modules (CM's and PM's). Each module is a conventional micro computer system. The connective structure of TOPSTAR is a locally connected bipartite graph. The control right belongs to each CM (which keeps unenabled data), and distributed control mechanism is attained. The load balancing problem is solved by the free competition rule.

Five kinds of basic nodes are prepared for programming in the data flow graph language. Most Petri nets, involved in the Free-Choice Petri net subclass, can be realized by using these basic nodes.

System software is implemented. The overflow in data buffers is avoided by introducing inhibitors (of the Petri net). Plural data buffers are attached to each node, and tokens can get ahead of others to provide more pipeline parallelism. Several program structures (such as conditional branch, loop and recursion) are available.

TOPSTAR-II consisting of 24 microprocessors is constructed and working with some applications. They are parallel mergesort, simulation of logical circuits, some arithmetic calculation, etc. Actually measured result in the case of parallel merge-sort are also reported.

1. Introduction

The data flow concept is a principle well suited to control a highly parallel system. Locality in the system control mechanism as well as mutual dependency among process executions can be attained [1]–[4]. It is necessary to construct a real machine and gain wide experience, in order to investigate the practical availability of a data flow machine.

Data flow machines are considered to be multi-processor systems, in which each processor has instruction level data flow control mechanism [1]. We introduce procedure level data flow processing as the first step, where one node of a data flow graph corresponds to one procedure. Each procedure (i.e. inside program of each node) may be written as a conventional program, and executed in the conventional manner. However all high level (inter-procedures) relations are controlled and executed as a data flow program.

Procedure level data flow processing can extract less parallelism than instruction level. But it is suitable for multi-microprocessors system to utilize conventional LSI processors, and it is also favorable from the view point of the time ratio of transfer overhead period (it is often dominant) to execution period [5]. It is also possible to replace each conventional LSI processor with an instruction level data flow machine.

TOPSTAR was designed and constructed as a procedure level data flow machine. The system hardware is composed of two kinds of modules which play the roles of 'place' and 'transition' in Petri net [6], which has good correspondence with a data flow graph. Each data flow processing node, which is a procedure, is separated into 'place' and 'transition' and they are executed independently on two kinds of modules. The most important feature of the data flow system is the connective structure, which decides the communication cost and efficiency. Inter-module connections of TOPSTAR are overlapped so that the system flexibility is increased. Connective range is restricted among modules so that highly parallel machine may be constructed easily. DMA (Direct Memory Access) is used as a high speed communication mechanism with minimal overhead, in order to install direct high speed data transfer between main memories.

A data flow graph language is used in TOPSTAR. It is designed to make use of programming structures (such as conditional branch, loop, recursion etc.). They can be implemented by using five kinds of basic nodes in the system software of TOPSTAR. The current system of TOPSTAR supports most graphs in Free-Choice class of Petri net.

In implementation, data buffers should be controlled not to overflow. This problem is solved by introducing 'inhibitors' of Petri net. In order to extract more parallelism in the pipeline effect, a flow control mechanism is implemented, such that tokens may get anead of others in a pipeline.

TOPSTAR-II consisting of 24 micro-processors is constructed and working with some applications. The

*Faculty of Engineering, University of Tokyo, Hongo, Tokyo 113, Japan.

parallelism in the case of parallel merge-sort is investigated.

2. Hardware Structure

The system is composed of two kinds of modules: PM (Processing Module) and CM (Communication, Control Module). PM and CM correspond to 'transition' and 'place' in Petri net, and they are constructed with microprocessors (Z-80) and peripheral LSIs such as an interrupt controller and a DMA controller.

One of the design problems of data flow multiprocessors exists in the communication and connection method between processors in terms of efficiency and adaptability to applications. It is desirable for the hardware structure to be similar to software structure. In our system, each PM is connected to many CMs, so that hardware has a dynamic structure and adjust itself to the structure of software by selecting one of the connective lines dynamically. Overview of system hardware is shown in Fig. 2-1.

2.1 DMA Connection

DMA connection is used to communicate between PM and CM at high speed. It is used because data transfer rate is a dominant factor in data flow processing, and block data transfer of procedure level processing is suitable for DMA. Signal lines of a couple of DMA controllers (Intel 8257) are connected directly as shown in Fig. 2-2. Data are transferred from memory to memory in distributed control. Common clock is used so that synchronization may be needed only at the beginning of a block transfer.

2.2 Dynamic Structure

As in Fig. 2-1, every PM is connected to several CM's, and every CM to several PM's, in connection such as bipartite graph of two kinds of modules. Each PM is

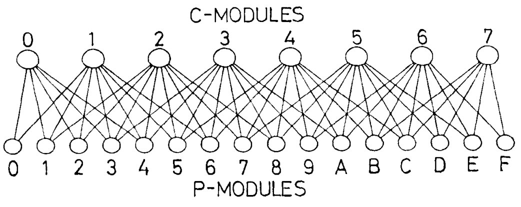


Fig. 2-1 A system overview.

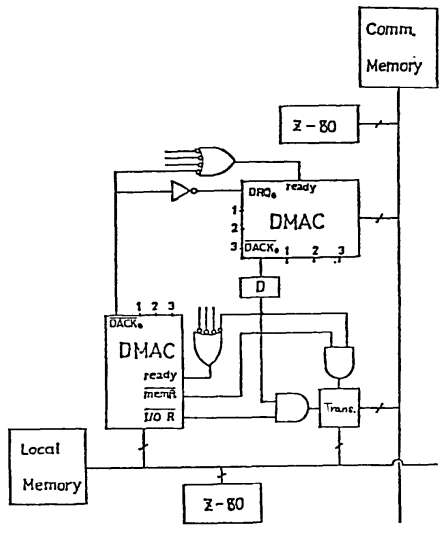
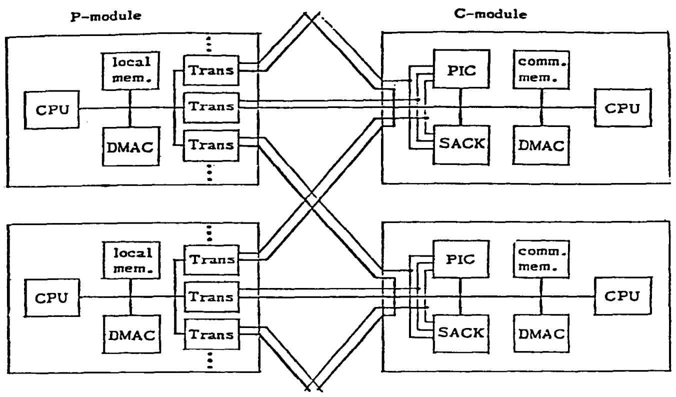


Fig. 2-2 DMA connection.



DMAC:	DMA Controller
Trans:	Transceiver
PIC:	Programmable Interrupt Controller
SACK:	Session ACKnowledge register
comm.mem.:	Communication Memory

Fig. 2-3 Inter-modules connection.

dynamically employed by one of the connected CM's, and each CM can employ as many PM's as necessary. Access racing problems are solved by using interrupt controllers (PIC: Intel 8259) and SACK (Session ACKnowledge) register. One line is selected by rotational priority and is connected according to the value of SACK register. Control rights belong to each CM, and distributed control system is realized. The connection diagram is shown in Fig. 2-3.

2.3 TOPSTAR

A prototype system "TOPSTAR-I" composed of three PMs and two CMs followed by a practical system "TOPSTAR-II" composed of 16PM's and 8CM's (each CM is connected to 8PM's) were completed and are working with system software on practical applications (mentioned later). The system is designed so that it can be easily expanded by plugging in new modules. The performance is being analyzed using a simulator written in GPSS.

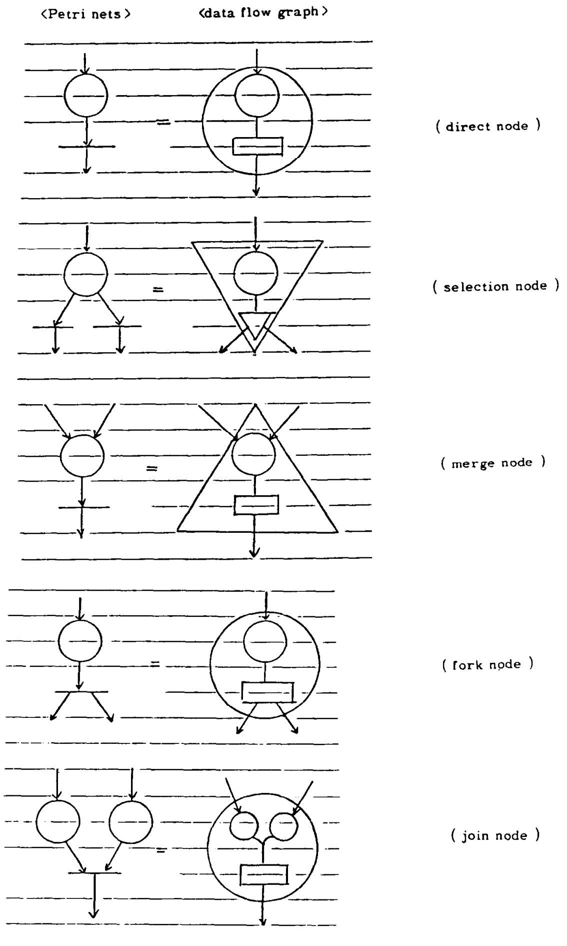


Fig. 3-1 Five basic nodes.

3. System Software

The control of data flow processing is quite simple, which can only monitor whether each process has taken all necessary data. But in implementing it on a real machine, various constraints occur because of limited resources, for example load balancing due to a finite number of processors, and overflow or deadlock problem of memory buffer used to maintain unenabled data.

3.1 Five Basic Nodes

The flow of tokens in a data flow graph can be represented as a Petri net. A node of a data flow graph in TOPSTAR is a procedure. Therefore it is important to control the flow of data, but not to prepare basic instructions.

In TOPSTAR system, five basic nodes are prepared from the view point of the flow control. Using these five nodes, most data flow graphs, which corresponds to Free-Choice Petri net class, are constructed. Figure 3-1 shows the five basic nodes and corresponding Petri nets.

3.2 Free Competition Control

Using the overlapped connection, we can change connective relations dynamically. In order to do that, each PM may search for an enabled process queued in CM's. Then many PM's gather around a busy CM (where many enabled processes exist) automatically, and the optimal structure is constructed.

A PM can request one enabled process by sending a DEQ command to a CM. If the CM has an enabled

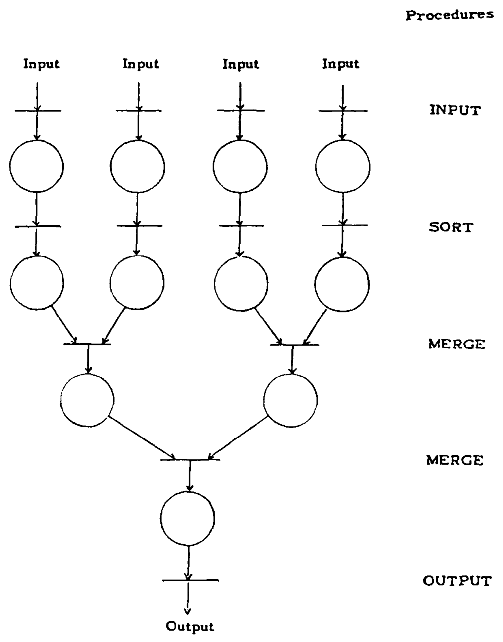


Fig. 3-2 Petri net of parallel merge-sort program.

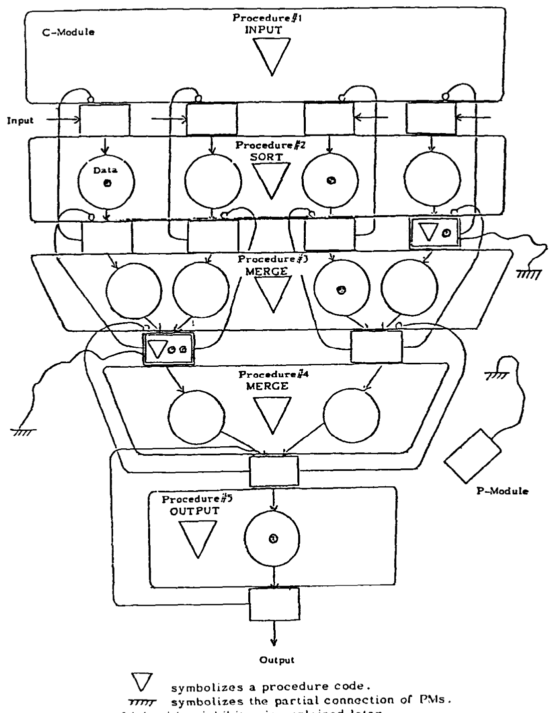


Fig. 3-3 Snapshot on TOPSTAR.

processes it then transfers procedure (if necessary), data and destination 'place' name to the PM by DMA. If not, PM may request it by another CM.

When a PM has finished its process, the PM sends output data, using an ENQ command, to the CM which contains the destination 'place'. While each CM is supervising all 'places' contained internally.

A parallel merge-sort program, developed as a test program on TOPSTAR-I, is shown to illustrate the behaviour of the system. Petri net corresponding to the program is shown in Fig. 3-2, and a snap shot of the execution on TOPSTAR-I is shown in Fig. 3-3.

In Fig. 3-3, a 'bar' in Petri net becomes a 'box' where an adequate PM goes in and executes the process. And a 'place' means a buffer in CM. In this snapshot, one sort node and one merge node are active. Tokens in 'places' are unenabled data. And one PM is idle.

3.3 Flow Control

In implementing, buffers (which keep unenabled data in CM) should be prevented from overflowing. It means adding a new condition to the enable rule "at least one buffer of the next nodes should be empty". This corresponds to introducing inhibitors in Petri net. The inhibitor is implemented as a V-OP command (see Fig. 3-4). The number of empty buffers of the next nodes are controlled by semaphores.

To execute data flow programs efficiently, constraints

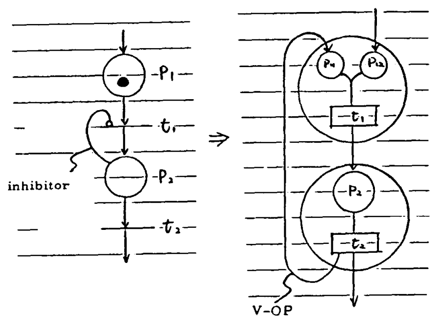


Fig. 3-4 'Inhibitor' is implemented as a V-OP command.

other than data dependency should be excluded. On the other hand, loop and recursion are desired in program structures. These requirements are implemented, thus racing and deadlock problems are solved.

⟨Outrunning problem⟩

In the usual pipeline processing, data cannot enter a processing element until the previous data has gotten out of it (FIFO control). This is a constraint other than data dependency. In order to gain more parallelism, we used multiple token places and allowed flowing data to get ahead of others.

To identify each data, a serial number (it is equivalent to 'coloured token' [7], [8]) is attached to them at the input ode. But with finite token places, there may be a deadlock at a join node in outrunning circumstances (see Fig. 3-5). It can be avoided by reserving one place for the data that have the earliest serial number. These

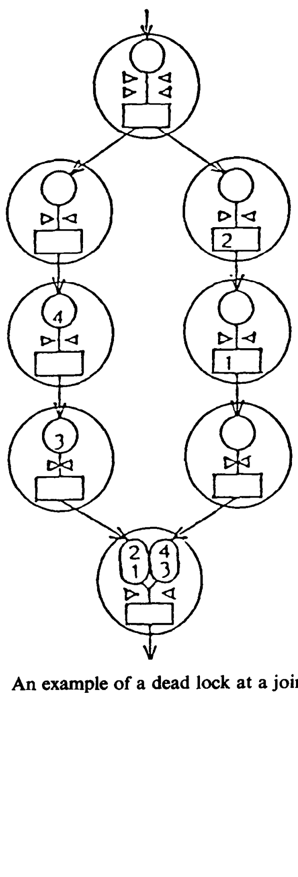


Fig. 3-5 An example of a dead lock at a join node.

controls are performed by sending V-OP commands from PMs' to CMs'.

⟨Program structure⟩

Loop and recursion are allowed as program structures. They can be constructed by combining a selection node and a merge node.

Loop

Figure 3-6 is an example of a conditional branch and a loop. An input can enable the node and one output is selected. The selection is decided inside the node by the selective procedure itself. In these cases, the flow should be controlled at the outside of the body (see Fig. 3-7).

Recursion

Stack memory is usually used in each node for recursive control. However, in TOPSTAR, in order to make each node nonhistorical, each data keeps its own stack to record the history (the list of node names it passed). The data format is shown in Fig. 3-8.

A recursive graph is converted to the graph as follows using PUSH and POP nodes:

1. Make PUSH and POP nodes at the entrance and exit of the graph.
2. Connect the recursive call links to the PUSH node.
3. Connect the recursive return links to the output links of the POP node.

When data enter the PUSH node, it pushes down the return node name into the header of the data. When data enter the POP node, it pops out the node name and

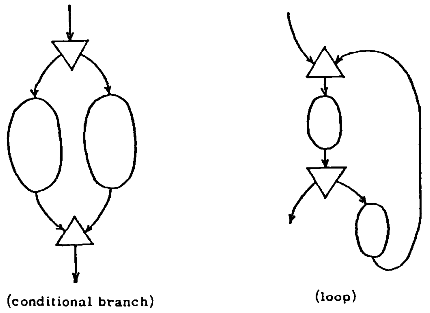


Fig. 3-6 Examples of program structures.

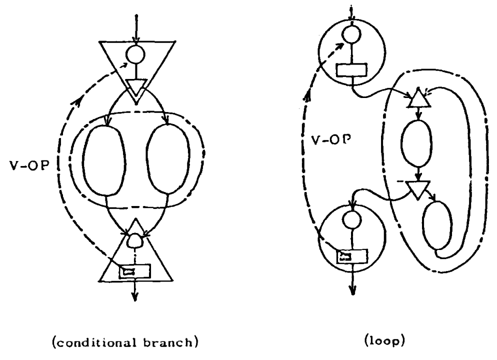
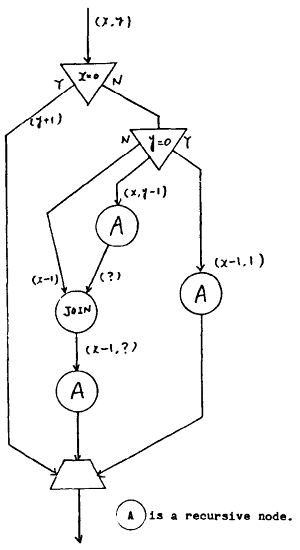


Fig. 3-7 Flow control of program structure.

SN	SP	d1	d2	...	dsp-1	data body

SN: Serial number.
SP: Depth of the stack.
di: Node name the data passed.

Fig. 3-8 The data format.



```
A(x,y) = if x=0 then y+1
    else if y=0 then A(x-1,1)
    else A(x-1,A(x,y-1))
```

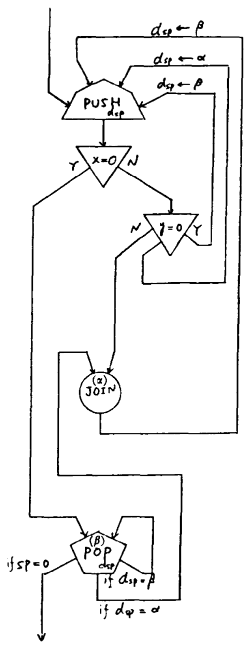


Fig. 3-9(a) Recursive definition of Ackermann function
Fig. 3-9(b) The result of the conversion.

selects one of the output links according to the name. Ackerman function is defined recursively as in Fig. 3-(a). The graph of Fig. 3-9(a) is converted to the graph of Fig. 3-9(b), which can be executed on TOPSTAR.

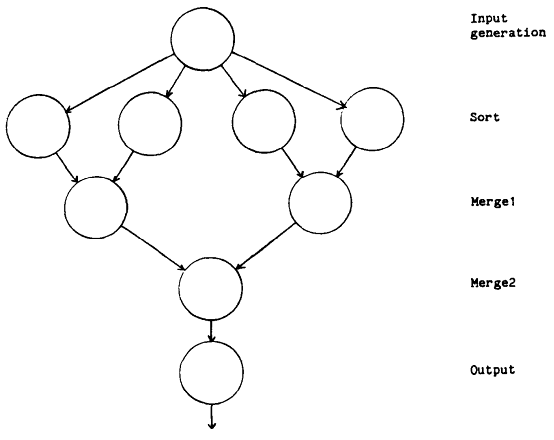


Fig. 4-1 Parallel merge-sort program.

4. Applications and Estimations

TOPSTAR system was originally developed for parallel processing of pattern recognition or artificial intelligence. Application for the recognition of printed Chinese characters was examined. Detail was shown in another paper [9].

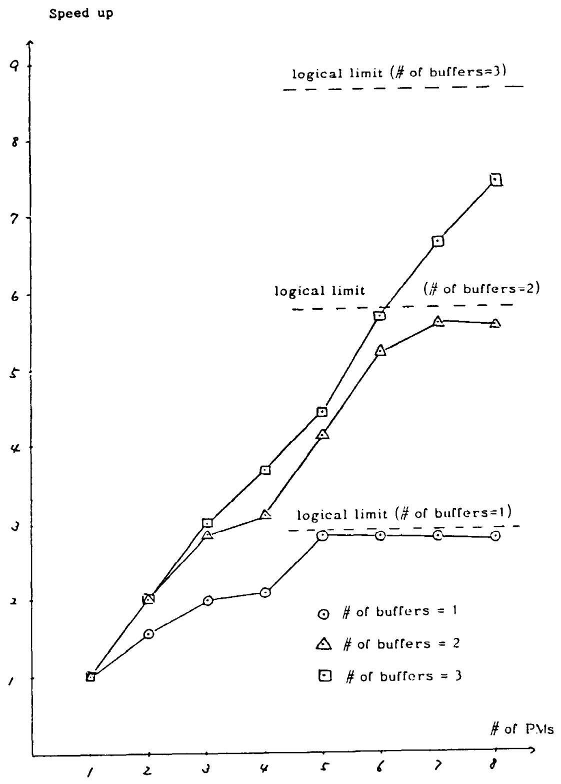
Several application programs are working. We present the result about parallel merge-sort program (Fig. 4-1). Mean execution times of these nodes are shown in Table 4-1. Serial execution time becomes 5.515 sec. Figure 4-2 shows the processing speed up by increasing the number of PMs'. It also estimates the pipeline effect of plural data buffers. Logical limit is decided by the critical node (which has the maximum execution time) of the data flow graph. System overhead (i.e. command exchanging time) in this case is shown in Table 4-2.

Table 4-1 Mean execution time of each procedure.

Procedure name	Input generation	Sort	Merge 1	Merge 2	Output
Exec. time (sec)	1.912	0.816	0.083	0.171	0.0017

Table 4-2 System overhead for each command.

Command	DEQ	ENQ	V-OP
Time (msec)	2.3	1.3	0.8



Speed up = Serial execution time/Parallel execution time

Logical limit = # of buffers * Serial execution time/Critical node execution time

Fig. 4-2 Speed up of the pipeline in the case of parallel merge-sort.

5. Conclusion

Dynamic structure data flow machine TOPSTAR consisting of multi-microprocessors was designed and constructed. It is composed of two kinds of modules (CM and PM) which play the roles of 'place' and 'transition' in Petri net. Separation of each processing to 'place' and 'transition', and the overlapped connection between modules made this system highly flexible.

Procedure level data flow processing system was implemented on TOPSTAR. Data driven mechanism is realized by exchanging three commands (DEQ, ENQ and V-OP) between CMs' and PMs'. With the flow control program, data can get ahead of others in pipelining to attain higher parallelism. Loop and recursion are used as program structures. Some applications are successfully implemented, and the actually measured result about parallel merge-sort program is reported.

References

1. Rumbaugh, J. E. A Data Flow Multiprocessor, *IEEE Trans. on Computers*, **C-26**, 2 (Feb. 1977), 138–146.
2. Davis, A. L. The Architecture and System Method of DDM1: A Recursively Structured Data Driven Machine, Proc. of the 5th Annual Symposium on Computer Architecture, *Computer Architecture News*, **6**, 7, Apr. (1978), 210–215.
3. Keller, R. M. et al. A Loosely-Coupled Applicative Multi-Processing System, *AFIPS, Proc. of the NCC*, 1979, 861–870.
4. Arvind and K. P. Gostelow. A Computer Capable of Exchanging Processors for Time, *Information Processing* 77, North Holland (1977), 849–853.
5. Denning, P. J. Operating Systems Principles for Data Flow Networks, *Computer*, **11**, 8 (July 1978), 86–96.
6. Peterson, J. L. Petri Net, *ACM Computing Survey*, **9**, 3, 223–252.
7. Dennis, J. B. First Version of A Data Flow Procedure Language, *MIT/LCS/TM*-61 (May 1975).
8. Dennis, J. B. and D. P. Misnus. A Preliminary Architecture for A Basic Data Flow Processor, *Proc. of the 2nd Annual IEEE Symposium on Computer Architecture* (Jan. 1975), 126–132.
9. Suzuki, T. and T. Moto-oka. Pipeline SAMD Machine and Its Applications to Recognition of Printed Chinese Characters, *Proc. of the 4th Int. Joint Conf. on Pattern Recognition* (1978), 1082–1086.

(Received October 3, 1980: revised August 12, 1981)