Invited Paper

Dataflow Computer DFNDR-2 and its Extension

MASAHIRO SOWA*

Dataflow processing has the special advantages of freedom from side-effect and the optimal exploitation of the inherent parallelism of programs. It also has shortcomings such as poor efficiency in executing serial programs, non-compatibility with conventional computers, non-flexibility and poor efficiency in manipulating data structures. To avoid these shortcomings, a program counter, the mechanism of von Neumann processing and the mechanism of parallel control flow processing are introduced to dataflow computers. This paper describes methods of introducing such mechanisms into the dataflow computers. This introduction could be done with almost no increase in the hardware cost.

1. Introduction

Recently dataflow computers have attracted many researchers because they exhibit some desirable features which von Neumann computers lack. Basic studies of dataflow schemes have been promoted by many research organizations and some have been built or are currently under construction (*e.g.* [1-11]). Pure dataflow processing has the special advantages of freedom from side-effects and the optimal exploitation of the inherent parallelism of programs.

On the other hand, the dataflow system suffers shortcomings such as poor efficiency in performing serial processing, low-ability to execute von Neumann processing, non-flexibility resulting from performing programs only in accordance with direct dependence among data [17], poor efficiency in manipulating the data structure [13], etc. There is always a certain amount of serial processing in a dataflow program. In serial processing, dataflow performance remains unequal to that of von Neumann computers. The data structure should be treated in serial because of the unsynchrony principle [12]. In this case the relatively slower execution of the serial part(s) of a dataflow program may negate the speed-up obtained in the execution of the concurrent parts(s). In programs like, for example, searching, sorting and serial I/O processing, serial processing may be preferred for simplification or high speed processing. But it is almost impossible to rewrite a parallel program into a serial one.

Although the control flow processing has a drawback in which it suffers side effects, it has great flexibility such as ability of writing many kind of programs to solve one problem, and capability of manipulating data structures in parallel. This is because the processing mechanism is not directly affected by the dependency existing among data. The principle underlying the currently most popular von Neumann computer system is based on the concept of control flow processing [12]. That is, it is a serial control flow processing by means of a program counter. The great flexibility is one of reasons why a von Neumann computer is so popular in spite of its serial processing property.

To avoid the shortcomings, mentioned above, of data flow computers, we proposed to introduce a program counter to speed up its serial processing. Von Neumann processing is introduced mainly to get compatibility with conventional computers and also to speed up its serial processing, and parallel control flow computation mechanism to increase flexibility into dataflow computers.

This paper describes how to introduce them into a dataflow computer.

2. Dataflow Computer DFNDR-2

In a dataflow computer DFNDR-2, which is a virtual machine derived from DFNDR-1, built in our university on an experimental basis [10], an **actor** or instruction is constructed as an **actor packet** and a **token** or data is constructed as a **token packet** (TP). To represent these in detail, BNF syntax is used with brackets [ ] indicating one or more repetitions of enclosed item(s). The actor packet consists of a function and one or more output arcs. The output arc also serves as the input arc of the destination actor.

⟨actor packet⟩ :: =⟨function⟩[⟨output arc⟩]

⟨output arc⟩ :: =⟨destination actor name⟩
⟨destination actor input arc number⟩

The ⟨destination actor name⟩ (DNA) is the name of the actor to which the resultant data or token is to be sent.

*Department of Electrical Engineering and Computer Science Nagoya Institute of Technology, Gokiso, Syouwaku 466, Nagoya, Japan.

It is therefore the address in memory where the actor is to be stored. The ⟨destination actor input arc number⟩ identifies which input arc of the destination actor receives the token. The resultant token is sent to a so called token memory as a ⟨resultant token packet⟩, which consists of one resultant token, a color and an output arc.

⟨resultant token packet⟩ :: =⟨output arc⟩
⟨color⟩⟨token⟩

The color shows the instance of its program.

In the token memory, the token packet is constructed from ⟨resultant token packet⟩s which have the same ⟨destination actor name⟩ and ⟨color⟩. Thus, the token packet consists of a ⟨destination actor name⟩, ⟨color⟩ and one or more tokens.

⟨token packet⟩ :: =⟨destination actor name⟩
⟨color⟩
[⟨destination actor input arc number⟩
⟨token⟩]

A ⟨token packet⟩ is said to be completed when it receives all the necessary token(s) for firing its associated actor. It is then called a **complete token packet** (CTP).

The DFNDR-2 dataflow computer consists of five major components: 1) a **token memory**(TM), 2) **processing units**(PU's), 3) an **arbitration network**(ARN), 4) a **memory**(M) and 5) **broadcast bus**(BB) as shown in Figure 1.

Token memory (TM) is a relatively small multiport and associative memory which stores ⟨token packet⟩s. Memory (M) is an interleaved multiport memory which holds data structures and the dataflow program in the form of ⟨actor packet⟩s. Arbitration network (ARN) resolves memory contentions and connects processing units to the memory. A broadcast bus(BB) drown in broken lines is used in section 4.

An actor is executed by one processing unit as follows. An available processing unit fetches a complete token packet(CTP) from the token memory **(CTP fetch cycle)**. Then, using the ⟨destination actor name⟩ of the complete token packet, it fetches an appropriate ⟨actor packet⟩ from the memory **(Actor packet fetch cycle)** and executes the specified ⟨function⟩ on the data in the

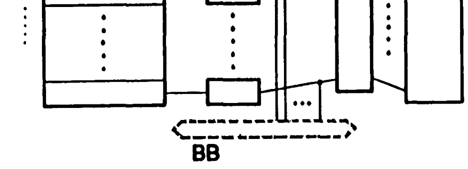


Fig. 1 Structure of the computer.

complete taken packet. **(Execution cycle)**. After execution, resultant tokens are sent to the token memory as the ⟨resultant token packet⟩s. ⟨resultant token packet⟩s with the same ⟨destination actor name⟩ and ⟨color⟩ are combined and stored in the one word of the token memory. This will become a complete token packet **(update cycle)**. This combination is determined by the associative function of the token memory. And its process is similar in concept to the matching of tokens in other dataflow computers. The T_i in Figure 1 is the token which is to be sent to the "i"th input arc of the destination actor. As described earlier, in this architecture all data items are passed to the actors via the token memory, and the actor is allocated dynamically to the processing unit. So no concern is needed for allocating a program when loading it.

3. Introduction of a Program Counter into the Dataflow Computer

Even in serial execution in dataflow computers, explicit expressions are required to be used for transferring data between instructions and for link manipulation, which makes serial processing in a dataflow computer slower. In serial processing, however, the order of execution is deterministic because only one actor is firable at any given time. It is possible, therefore, to simplify the operations by introducing two additional features into the dataflow computer: a) The exclusive allocation of one processing unit to each serial part of a dataflow program, b) The storage of actors in execution order in the memory.

From a), one of the resultant tokens can now be passed to the next actor via the processing unit rather than the token memory. This can result in faster token exchange. From b) which is popular in von Neumann computer, the memory address of the next actor, the ⟨destination actor name⟩, can be calculated by adding the word length of the executing actor to its ⟨destination actor name⟩ (except for a BRANCH-like-instruction). This permits the omission of one of the arcs between the actors. An increase in speed is highly probable because one of the link manipulations is now dispensable.

To implement this mechanism, it is necessary to add a program counter and an accumulator to each processing unit, and to store the actors of the serial parts of the programs in execution order in the memory. This makes it possible to send one of the resultant tokens to the next firable actor via the accumulator. The next firable actor is found by the program counter as in a von Neumann computer. The fetching of an ⟨actor packet⟩ moves the program counter on by the appropriate number of increments. It should be emphasized here that a dataflow computer, even with a program counter and accumulator, is nevertheless distinct from a von Neumann computer. While a dataflow computer with a program counter executes serial dataflow programs on a data driven principle, a von Neumann computer executes serial control flow programs. That is, their execution model is the same but computation model is different. The former has no side effect, but the later has.

The choice of the next firable actor is determined by the program counter, in contrast to the complete token packet method shown in section 2. In effect, the addition of the program counter update operation allows the omission of the operation fetching ⟨destination actor name⟩. The number of iteration required for token fetch, output arc fetch and ⟨resultant token packet⟩ storage is decreased by one.

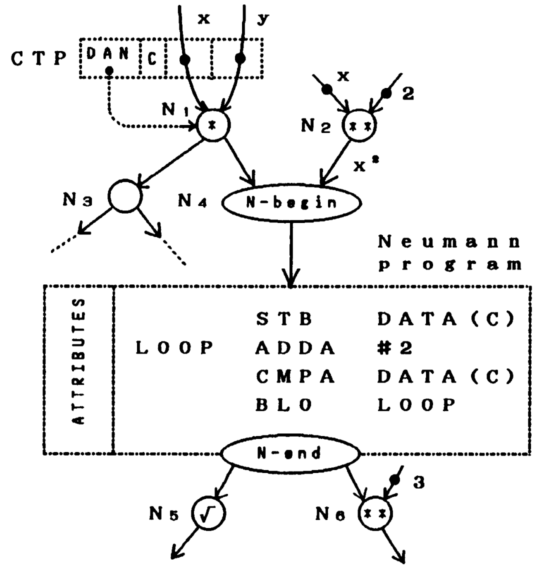
Switching between the conventional and modified execution modes occurs when the code is interpreted, in accordance with information embedded in the ⟨function⟩ code.

Simple analysis based on the constant values actually obtained from the dataflow computer DFNDR-1 machine, which was built from TTL with MOS memory, says that this method accelerates the execution of serial processing by as much as 20–50% [11].

4. Introduction of von Neumann Processing

By adding von Neumann processing ability to the dataflow computer, the computer will have great flexibility, compatibility with current computers and the ability to execute serial programs at high speed. It implies that together with dataflow processing, the computer can execute a large number of von Neumann programs which have been developed.

In order to implement the von Neumann mechanism, a program counter and accumulator *A*, *B* and *C* are added to each processing unit.



```
LOOP    STB     DATA (C)
        ADDA    #2
        CMPA    DATA (C)
        BLO     LOOP
```

Fig. 2 Date flow program including von Neumann one.

Figure 2 shows an example program where both von Neumann and dataflow processing exist, where a complete token packet(CTP) is superimposed. **N-begin** in the figure is an instruction indicating that after this instruction, von Neumann program should be executed. That is, when the processing unit fetches an ⟨actor packet⟩ from the memory and its function is **N-begin**, the processing unit puts the ⟨destination actor name⟩ of the ⟨output arc⟩ into the program counter, then begins von Neumann processing. While **N-end** is an instruction to indicate the end of execution of a von Neumann program. When **N-end** instruction is fetched, the processing unit begins dataflow processing by using data in the accumulator and output arc(s) of **N-end**. In this program, the input data to the left input arc of **N-begin** is given to accumulator *A* and the input data from the right is given to accumulator *B*. The content of accumulator *A* is output to the output arcs of **N-end**. To calculate the minimum value of $x{*}y+2n$ not less than x^2, the LOOP body of the von Neumann program is repeated *n* times. It is sure that the needed data are fetched and stored from/to the memory(M) during von Neumann execution.

In the dataflow processing, color is used to show the instance of its program. Coloring is performed when the procedure is called. When the execution of the procedure is completed, coloring in the reverse direction is performed.

In the case where the procedure contains a von Neumann program, there must be prepared a mechanism to handle the coloring for von Neumann program. That is, a certain work memory area for the von Neumann program must be prepared for each call. In other word, a work memory area must be prepared for each instance of the von Neumann program. The work memory area, which stores the data of the program, is provided for each token color given in the complete token packet to the **N-begin** actor. In other words, when the complete token packet is input to **N-begin** actor, a work memory area(WMA) correspond to the color of the complete token packet is given by the memory manager and the beginning address of the work memory area is put into the accumulator *C*, then the processing unit begin executing von Neumann program by using accumulator *C* as a base register, which is popular in, for example, the IBM 370 computer. The work memory areas is released when **N-end** instruction is executed.

This introduction makes it possible for the computer to do parallel processing together with von Neumann processing. A simple analysis says that the serial processing of von Neumann computers is about 3.4 times faster than that of dataflow computers with no program counter [12]. In general, with VLSI technology, it also seems unlikely that cost would be prohibitive.

5. Parallel Control Flow Processing

In the dataflow processing, data flows in a program. In contrast, in the control flow processing, the token indicating "control of execution" which is termed **control token** flows in the program. With the phrase "control of execution," it is intended to mean "permission of execution for an instruction or an actor" so that only upon arrival of the control token(s) at a instruction, the associated execution is allowed to be initiated. It should be noted that in case of data flow processing, data represents value to be processed and at the same time serves for the function of the control token.

Figure 3(a) illustrates a control flow program for calculating $d=(ab+bc)(a-b)$. At each of the actors, the function or instruction described in the actor is executed upon arrival of the control token at the associated input arc. Upon completion of the execution, the control token is output on the associated output arc(s). In Figure 3(a), it is assumed that executions at the actors *N*3, *N*4 are in the state available for execution. Symbols ***M***, ***S*** and ***A*** represent, respectively, multiplication, subtraction and addition, and ***x***, ***y***, ***z***, ***a***, ***b***, ***c*** and ***d*** represent variables. Values of the variables are stored in the memory the same as in von Neumann computers. The instruction ***M b***, ***c***, ***y*** in actor *N*2, for example, means that data stored in the memory at the addresses ***b*** and ***c*** are to be read out and to be multiplied with each other, then the resultant data is to be stored in address *y* of the memory. In its expression, the control flow program differs from the dataflow program in that an instruction contains designators or addresses for the data. The illustrated program suggests the possibility of the processing at the actors *N*1, *N*2 and *N*3 being executed in parallel.

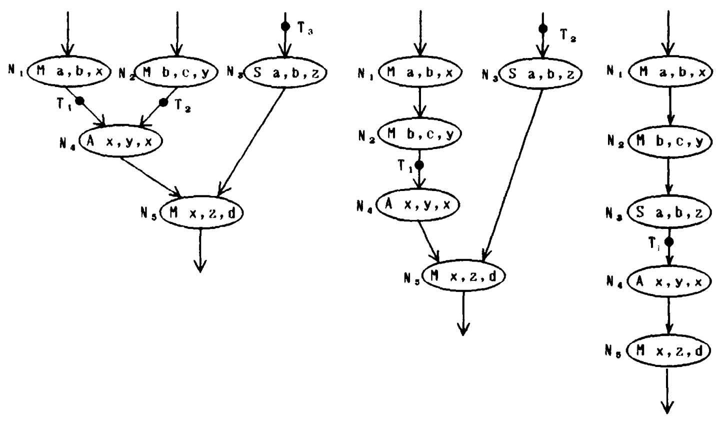
In dataflow processing, the data must necessarily be prepared by the immediately preceding actor. In contrast, in case of the control flow processing, data may be prepared at any actor so far as they are located upstream of the actor whose content is to be executed. In this sense, it can be said that the control flow processing has no direct dependence on the data. This is the main reason why the control flow processing has great flexibility. Since the control flow processing has no direct data dependency, the program in Figure 3(a) can be rewritten as shown in Figure 3(b) or Figure 3(c). It is noted that the program in Figure 3(c) is a serial control flow program which is very similar to the von Neumann one. In fact, if the program would be rewritten again so as to be executed by using a program counter and accumulators, it becomes a von Neumann program.

The ⟨actor packet⟩ of the dataflow computer is modified to execute the control flow programs as follows.

⟨actor packet⟩ :: =⟨function⟩⟨operand1⟩
⟨operand2⟩⟨operand3⟩
⟨# of arcs⟩[⟨output arc⟩]

where ⟨operand1⟩, ⟨operand2⟩, ⟨operand3⟩ represent the memory address or data and ⟨# of arcs⟩ represents the number of ⟨output arc⟩s. Since the control token is used for indicating its "presence" or "absence", each token of the ⟨token packet⟩ may be of a single bit. That is, just one bit of the ⟨token⟩ in the ⟨token packet⟩ is used for the control token.

The control flow processing begins when the ⟨function⟩ of an ⟨actor packet⟩ is for the control processing. In the control flow processing, only the operation of the **Execution cycle** is different from a dataflow one. In the



(a) Control flow program

(b) Control flow program with reduced parallelism

(c) Serial control flow program

Fig. 3 Control flow programs d=(ab+bc)(a−b).

execution cycle, the processor reads out the data designated by the effective address made from the ⟨operand1⟩ and ⟨operand2⟩ from the memory and executes the function on the data. After completion of the execution, the resultant data is stored at the memory designated by the effective address of ⟨operand3⟩.

The work memory area should be prepared for the control flow processing by the similar reason in the von Neumann one. To prepare work memory areas for the control flow processing, a **work-memory-area table**(WMT) and a broadcast bus(**BB**) are added to the architecture of the dataflow computer. The work-memory area table is added in each processing unit and it consists of color(C) and work-memory-area pointer(WMP).

⟨work-memory-area table⟩ :: =⟨color⟩ ⟨work-memory-area pointer⟩

The broadcast bus is connected to each processing unit as shown in Figure 1. When a procedure including the control flow processing is called, a new color and the work memory area are given to the procedure instance. Then this new color and the beginning address of the work memory area are sent to all processing units via the broadcast bus. The color is stored in the ⟨color⟩ part of the ⟨work-memory-area table⟩ of each processing unit and the beginning address is stored in the ⟨work-memory-area pointer⟩ part. The effective address is determined by adding the content of the ⟨work-memory-area pointer⟩ to ⟨operand1⟩, ⟨operand2⟩ or ⟨operand3⟩. The color and work memory area are released when the procedure execution is completed. A simple analysis says that the speed of the parallel control flow processing is 32% slower than that of the dataflow one [17].

6. Conclusion

To overcome the shortcomings of the dataflow computer, the methods to implement a program counter, von Neumann processing and parallel control flow processing have been described based on the DFNDR-2.

It has been shown that by introducing a program counter and an accumulator, the execution time of the serial part of a dataflow program can be reduced by between 20-50%. It can be seen that the von Neumann concepts of a program counter and an accumulator could be usefully employed, even in a dataflow computer, to speed up serial processing.

Introducing von Neumann processing into the dataflow computer increases the flexibility of the processing and speed-up of serial processing. By this introduction, it becomes possible to execute tremendous number of conventional programs by this computer. This will realize a mixed processing of dataflow and von Neumann.

Since execution order of actors of the control flow processing does not directly depend on the data dependency, the control flow processing can enjoy correspondingly increased flexibility as compared with the data flow processing. Due to this flexibility, the data structure can be processed easily in parallel. Of course, the enhanced flexibility in turn may bring a side effect to the processing itself. Such influence, however, could be evaded by executing the control flow program translated from the dataflow program [15]. The point is that with almost no increase of the hardware cost, the computer will obtain flexibility, compatibility, and speed-up of serial processing.

To confirm the proposals experimentally, the mechanisms without the introduction of a program counter have been implemented in the real data flow computer DFNDR-1.

References

1. DENNIS, J. B. First version of a data flow procedure language, *Proceedings of the Symposium on Programming, University of Paris* (April 1974).
2. ARVIND, GOSTELOW, K. P. Some relationship between asynchronous interpreters of data flow languages, Formal Description of Program Languages, E. J. Neuhoid, Ed., North-Holland, New York (1977).
3. WATOSON, I., GURD, A. A prototype data flow computer with token labeling, *Proceedings of the AFIPS Conference*, **25** (1982).
4. SOWA, M., MURATA, T. A data flow computer architecture with program and token memories, *IEEE Trans. on Computers*, C-**31**, 9 (Sep. 1982).
5. AMAMIYA, M., HASEGAWA, R., NAKAMURA, O. and MIKAMI, H. A list processing oriented data flow machine architecture, *Proceedings of National Computer Conference, AFIPS*, 143-151 (1982).
6. HIRAKI, K., SHIMADA, T. and NISHIDA, K. A hardware design of the SIGMA-1—A data flow computer for scientific computations, *Proceedings of International Conference on Parallel Processing, IEEE*, (1984).
7. SUZUKI, T., KURIHARA, K., TANAKA, H. and MOTOOKA, T. Procedure level data flow processing on dynamic multimicroprocessors, *Journal of IPS Japan*, **1.5**, 1, 11-16 (1982).
8. NISHIKAWA, H., ASADA, K. and Terada, H. A centralized controlled multi-processor system based on the data-driven scheme, *Proceedings of 3rd International Conference on Distributed Computing Systems, 639-644 (1982).*
9. GAJISKI, D. D., PADUA, D. A., KUCK, D. J. A second opinion on data flow machines and languages, *Computer*, **15**, 2, (Feb. 1982).
10. SOWA, M., RAMOS, F. D., MURATA, T. Construction and structure of prototype data flow computer DFNDR-1 and subroutine implementation, *Proc. of IEEE the First International Conference on Computers and Applications*, Beijing China, pp. 490-497 (June 1984).
11. SOWA, M. An effect of introducing program counter into a dataflow computer, *IECE Japan, Proceedings of 1986 Dataflow* Work shop, pp. 33-38 (May 1986).
12. SOWA, M. Execution mechanism of von Neumann program in dataflow computer, *IECE Japan, Tras.* J-**67**, 5 (May 1984).
13. SOWA, M. Control flow parallel computer architecture, *IPS Japan, Report of Computer Architecture Study Group*, 45-1, (March 1982).
14. TRELEAVEN, P. C., HOPKINS, R. P. RANLENBACH, P. W. Combining data flow and control flow computings, The *Comput. J.*, **25**, 2 (1982).
15. SOWA, M. Universal flow computer, *IPS Japan, Proceedings on Computer Architecture Work Shop in Japan'* 84, pp. 113-121 (Nov. 1984).
16. SOWA, M. and AOKI, T. Realization result of prototype control flow computer, *Annual convention IECE Japan*, **7**, pp. 1714 (March 1985).
17. SOWA, M., WATANABE, Y. Performance evaluation of parallel computer with cache memory and pipeline processing, *IECE Japan, Report of Computer Architecture Study Group*, EC**85**-36 (October 1985).

(Received November 2, 1987)