計算機アーキテクチャ 48－2
（1983. 3. 18）

コントロールフローパラレル
コンピューターアーキテクチャ

曽　和　将　容
（群馬大学　工学部）

ABSTRACT　A special feature of a control flow parallel computer (CFPC) is the presence of a so-called NODE-DRIVE REGISTER (NDR). It denotes the locations of executable instructions and corresponds to a program counter in a conventional stored program serial computer (CSPSC). By means of NDR it is possible to execute the instructions concurrently as well as nondeterministically without control complexity. The CFPC can execute not only control flow programs but also data flow ones.

I. Introduction

In recent years a number of research groups have proposed various data flow computer architectures [1] - [8] suitable for parallel processing. In contrast to these, the control flow parallel computer (CFPC) employed the concept of control flow used in a conventional stored program serial computer (CSPSC). What makes the CFPC different is the use of a so-called NODE DRIVE REGISTER (NDR). The NDR, corresponding to the program counter of CSPSC, is to store LINKAGEs and CONTROL TOKENs (CTs) for indicating, location(s) of executable instruction(s) and completion of the preceding instruction(s), respectively. The presence of the NDR makes it possible for processors to execute instructions concurrently and nondeterministically. The CFPC can execute also data flow programs because data flow processing is nothing but a data-restricted control processing. The NDR is expected to be relatively small in size because the control tokens occupy space in the NDR for a very short transient period and one control token can be constructed from one bit. The small scale NDR results in the simplicity of the interconnection network and the control circuit. A method to reduce the complexity of a multi-port main memory will be also described.

An instruction of a parallel computer program must have LINKAGEs to direct the next instruction(s) to be executed. Because one or more instructions may become executable after the completion of an instruction, and those instruction executions can not be directed by means of program counters as done in CSPSC.

II. Control flow program

A control flow program consists of nodes, arcs and control tokens. Instructions associated with the nodes are executed as

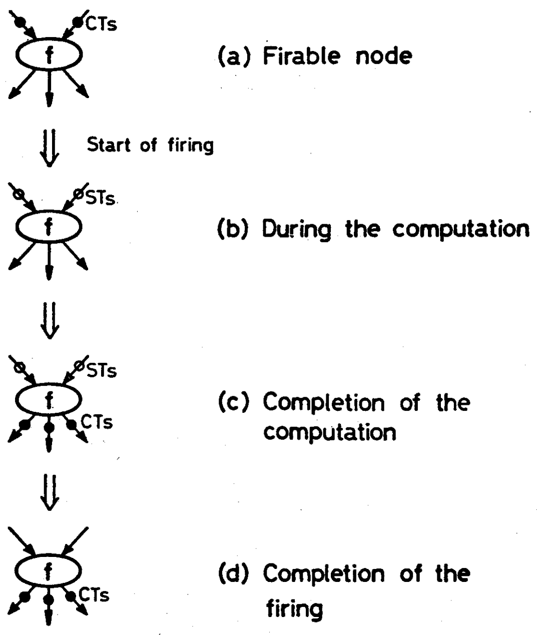
CTs
(a) Firable node
Start of firing
STs
(b) During the computation
STs
CTs
(c) Completion of the computation
CTs
(d) Completion of the firing

Fig. 1 Firing process of a node.

control tokens flow along arcs according to the firing rules as shown in Fig. 1. A node may fire whenever a control token is present on each of its incoming arcs (Fig. 1 (a)). When a node fires one control token is removed from each of its incoming arcs. As a replacement, a new token called SHELL TOKEN (ST) is put on each arc (Fig. 1 (b)) collectively. A control token is outputted on each of the outgoing arcs after the completion of the instruction or the function (f) execution (Fig. 1 (c)). Then the shell token on each of the incoming arcs is removed (Fig. 1 (d)). If it is impossible to output the control token on any outgoing arc because of too many tokens on it, STs are still removed from every incoming arc and a so-called RESULT TOKEN (RT) is put on it.

Fig. 2 shows an example of a control flow program for calculating $x=(-b+\sqrt{b^2-4ac})/2a$. The program is the same as the corresponding CSPSC flow-chart except for the parallel calculations and control tokens (CT_1, CT_2). The program and data (a, b, c and x) without the control tokens are stored in the main memory.

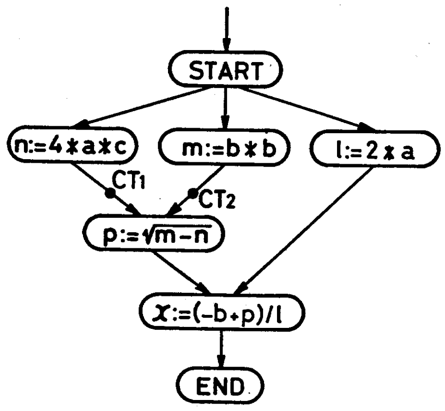

Fig. 2 Control flow program for computing $x=\frac{-b+\sqrt{b^2-4ac}}{2a}$.

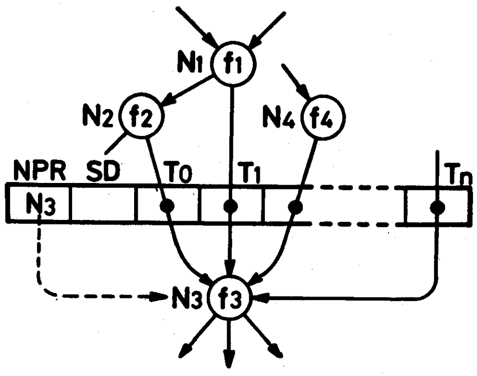

Fig. 4 Format of token packet.

III. Organization of a control flow parallel computer

The organization of a control flow parallel computer (CFPC) is shown in Fig. 3. The architecture is fundamentally that of a CSPSC. However, a node-drive register (NDR), plural processing units (PUs) and a multiport main memory (MM) are used instead of a program counter, one processing unit and conventional main memory, respectively. The NDR stores the control tokens, the shell tokens and the result tokens as so-called control token packets (CTPs), shell token packets (STPs) and result token packets (RTPs), respectively. Those token packets (TPs) consist of a node pointer (NPR), a status data (SD) and tokens (T0,...,Tn) which will be sent to the node indicated by the NPR, as shown in Fig. 4. A control token packet containing all the necessary control tokens is called COMPLETE CONTROL TOKEN PACKET (CCTP). The SD is used to distinguish the CTP, STP and RTP. A multi-port main memory (MM) stores control flow programs and data, and corresponds to the main memory of a CSPSC.

The NDR consists of a NODE POINTER MEMORY (NPM), a TOKEN INDICATOR & STATUS DATA (TS) and TOKEN COUNTERS (TC0,...,TCn) as shown in Fig. 5. The NPM and the TCs in a word of the NDR store the node pointer NPR and tokens of the token packet, respectively. Each TC is a counter which is incremented or decremented by storing or taking out a token respectively. The TS stores the SD of a token packet and the status of the word. The NDR is an associative and multiport memory to be accessed through each of the node pointers by plural processing units simultanously.

A CFPC program is stored as a set of so called NODE PACKETs (NPs) in the multi-port main memory. The NP consists of an INSTRUCTION or a FUNCTION (INS) and DATA ADDRESS(es) or DESTINATION NODE POINTER(s) (DA/DNP0,...,DA/DNPl) as shown in Fig. 6. The data address DA is a memory address into or from which data is written or read. The destination node pointer DNP is a linkage to the next node packet (NP) which is to be

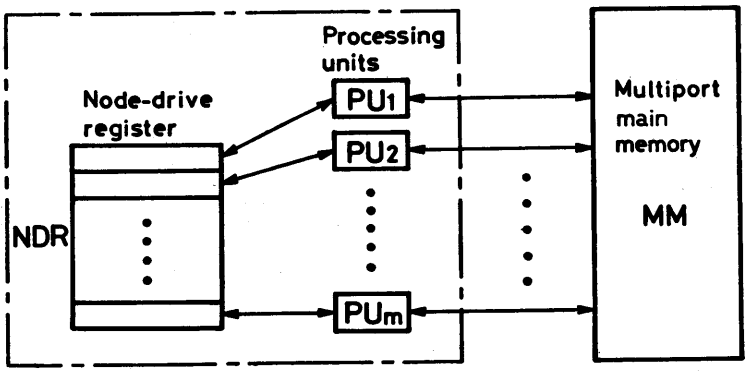

Fig. 3 Organization of the control flow parallel computer.

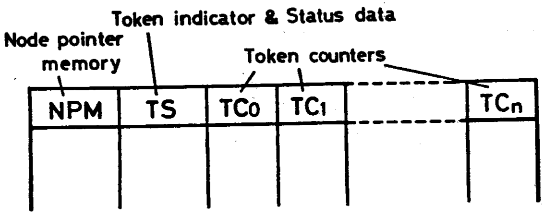

Fig. 5 Node-drive register (NDR).

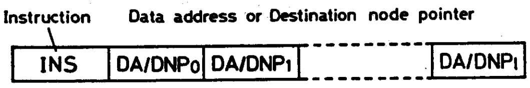

Fig. 6 Format of a node packet.

executed after the completion of the firing of the node. After starting of an NP-firing, the data denoted by the read-from DA(es) is(are) processed by the PU, and the result(s) is(are) written in the word(s) of the MM denoted by the write-into DA(es). Then the control token(s) is(are) sent to the NDR as CTP(s) using the DNP(s).

IV. Execution process

The execution process of a CFPC is essentially the same as that of the CSPSCs. The execution of a node or an instruction (Fig. 7) is divided into four stages: token packet fetch process, instruction fetch process, instruction execution process and NDR update process.

The following is a detailed execution algorithm for the architecture.

* Token packet fetch process
  1. A PU reads a CCTP, if there is any, from the NDR and writes "STP" in place of "CCTP" in the SD. Otherwise, the PU goes to step 1.
* Instruction fetch process
  2. The PU fetches from MM the NP pointed by the NPR of the CCTP read in step1.
* Execution process
  3. The PU executes the corresponding instruction or function of the NP.
* NDR update process
  4. After completion of the execution, the PU creates new CTPs using the DNPs in the NP fetched in step 2, and sends them to the NDR.
  5. The PU sends a delete signal for the STP to the NDR.
  6. When the NDR receives the new CTP from the PU, the NDR checks for CTPs, which were already stored, with the value of the NPR equal to that of the new CTP.
  7. If there is one, the NDR increments the TCs associated with the control tokens in the new CTP to be stored. If not, the NDR stores the new CTP in a vacancy

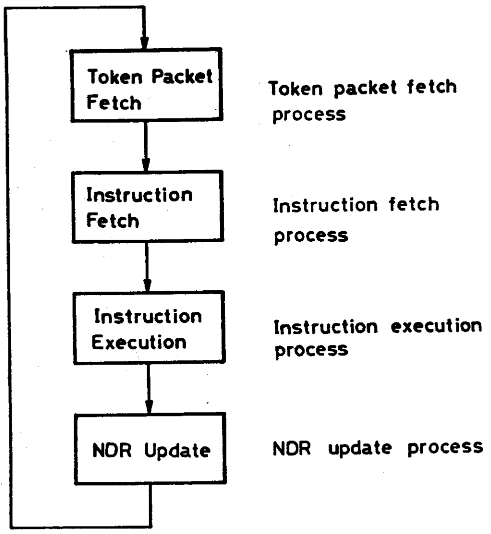

Fig. 7 Execution process.

  word of the NDR.
  8. When the NDR receives the delete signal for the STP in the step 5, the NDR removes the STP, then decrements all TCs of the word of the NDR in which the STP was stored.

In case the values of all TCs become zero in step 8, the word in which the STP has been stored becomes a vacancy. On the other hand an overflow of any TC may occur during step 7 thus, the PU is informed by means of an interrupt. Such overflow implies unsafeness of the execution. The PU receiving the interrupt changes the STP into RTP and writes the CTP(s), which could not send to the NDR for the next nodes, in the RTP. The sending of a result token resumes when the RTP is read in step 1 after such overflow ceases. That is, the PU begins the process from step 4 onwards and the PU sends a delete signal for the RTP instead of one for STP in step 5.

V. Reduction of complexity of the multiport main memory

The complexity of the multiport main memory MM can be reduced if the memory is divided two parts, a program memory (PM) and a data memory (DM), as shown in Fig. 8. The PM is a conventional memory which holds the control flow program, and is duplicated for each PU. Therefore the PM is a read-only memory for the PUs and is a write only memory for a host computer (HC) which writes control flow programs in it. The DM is also conventional memory and is divided into more than m banks to make a shared memory (the m is the number of PUs). A mutual exclusive control for each bank should be implemented but it is expected to be relatively small in

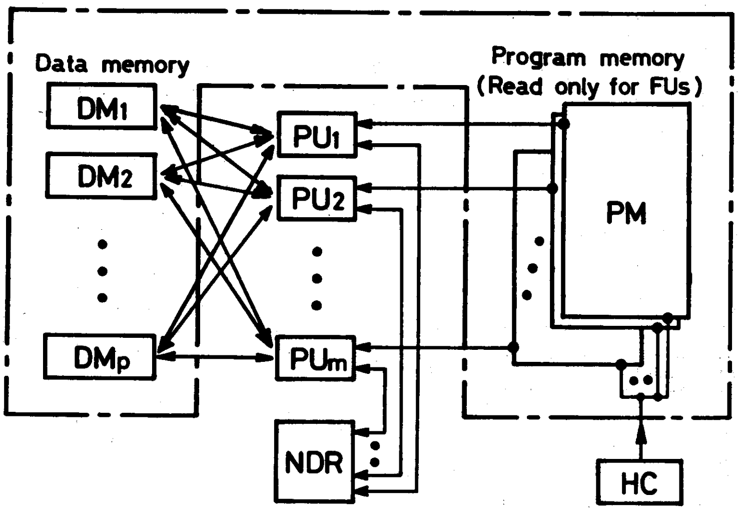

Fig. 8 Simplification of the MM.

size because the minimum number of the required banks to avoid data memory contentions is the number of the PUs.

VI. Conclusion

The concept of accumulators in a PU is not useful in this parallel computer because the accumulators are no longer common areas, as in the CSPSCs, but local areas.

It is easy for the control flow parallel computer to execute the data flow programs because the data flow processing is simply a kind of control processing directed by the availability of the required data. Therefore this computer is said to be a resource-driven [10] or combined driven [11] computer.

The NDR is quite similar to the token memory (TM) in the data flow computer proposed by us. However, as the NDR does not have to store data, it is expected to be simpler than the TM. And most of all technique developed in the CSPSC, for example, the subroutin call, the recursive call and the processing of structure, can be used because this computer operates in the concept of control flows. The presented architecture is one of more generalized architectures for parallel computers.

References

[1] J.B. Dennis and D.P. Misunas, "A preliminary architecture for a basic data flow processor", IEEE Proc. 2nd. Annu. Symp. Comput. Arch., 1975, pp. 126-132.

[2] A.L. davis, "The architecture of DDM1: A recursively structured data driven machine", Dep. Comput. Sci., Univ. Utah, Tech. Rep. UUCS-77-113, Oct. 1977.

[3] J. Rumbaugh, "A data flow multi-processor", IEEE Trans. Comput., vol. C-26, pp. 138-146, 1977.

[4] Arvind and K.P. Gostelow, "Data flow computer architecture: Reseach and goals", Dep. Inform. and Comput. Sci., Univ. California, Irvine, Tech. Rep. 113, Feb. 1978.

[5] A.D. Plas, D. Comte, O. Gelly, and J.C. Syre, "LAU system architecture: Parallel data-driven processor based single assignments", IEEE Proc. 1976 Int. Conf. Parallel Processing, Aug. 1976.

[6] J.R. Guard, I. Watson, and J.R.W. Glauert, "A multilayered data flow computer arhitecture", Dep. Comput. Sci., Univ. Manchester, England, July 1978.

[7] S.H. Yu and T. Murata, " Modeling and simulating data flow computations at machine language level", Proc. ACM Conf. Simulation, Measurement and Modeling of Comput. Syst., New York, Aug. 1979, pp. 207-213.

[8] M. Sowa and K. Hayakawa, "Procedure level data flow computer system -GMMCS-", Paper of Tech. Group Comput. Arch. 36-1, IIP Japan, 1979.

[9] M. Sowa and T.Murata, "A data flow computer architecture with program and token memories", IEEE Trans. Comput., vol. C-31, pp. 820-824, Sep., 1982.

[10] JIPDC, Proceedings of International Conference on Fifth Generation Computer Systems, pp. 57, March, 1982.

[11] P.C. Treleaven, R.P. Hopkins and P.W. Rantenbach, "Combining data flow and control flow computing", The Computer Journal, vol. 25, No. 2, 1982.