Design of Assembly Level Language for Control Flow Parallel Computer

曽和　将容　F. D. Ramos
（群馬大学　工学部）

SUMMARY

Capable of exploiting inherent parallelism in a program, the proposed parallel control flow computer is discussed. Potentially executable instructions will be known by examining the completeness of a control token packet which contains the already available control token(s) corresponding to a certain instruction or node packet. A control token packet is simple in structure and stored in a content-addressable memory called control token memory.

DATA FLOW and CONTROL FLOW REVISITED

To expedite program execution, researches about parallel processing are enthusiastically being performed. Of the proposed parallel processing computer architectures employing by availability control mechanism, data flow computer seems to be more popular than that of control flow because of its freedom from side-effects resulting, for example, in program verification facility. This peculiarity is attributable to an absence of shared memory location, better known as variable, in data flow programming. Data is passed directly between instructions and a separate copy, if necessary, is generated for each destination.

The aforementioned inseparable flow of data and control yields faster scalar arithmetic calculation. However, pure data flow computation is a failure with regards to manipulation of data structures because concurrent execution of logically parallel operation may result into data substructures that cannot be synthesized in order to obtain the desired structure. Consequently, what has been conceived to be a parallel operation has to be done in a sequential way. Therefore, to ascertain whether a logically parallel operation may be done concurrently or not becomes a must. Also, by value data mechanism is not commendable when the performance of, say, conditional expression does not make use of the separately generated data thus it has to be destroyed. This will surely occur in loop implementation. To alleviate these problems, non-data flow concepts have to be employed otherwise, general-purpose data flow computer will not be feasible. Such integration of non-data flow concepts results in its non-uniformity.

On the other hand, eventhough susceptible to side-effects, parallel control flow computer may turn out to be a better general-purpose machine. Like data flow, control flow utilizes the principle of by availability control mechanism. Sharing of computation results among the specified destinations is the cause of their differences. What is sent to the next

to-be-triggered instructions is simply a control token implying the availability of the corresponding argument(s). As a result, we may opt not to issue a control signal to specified next instruction(s) immediately after computation execution. What we have to be certain of is that once an instruction is activated, the necessary operands are all available. This will be meritorious when the degree of parallelism in a program may cause some sort of resource shortage or deadlock thus we have no option but to reduce it. In contrast to data flow programming, data initialization of a re-entrant procedure or loop in control flow programming does not require a separate specification because it may be easily included within the subroutine or loop's body, as in conventional Von Neumann computer, by means of "move immediate" instruction.

CONTROL FLOW PARALLEL COMPUTER ARCHITECTURE

The proposed control flow computer is organized as shown in Figure 1. The main memmory is modularized and 16-word interleaving is used so as to mitigate the problem of memory contention.

Control token memory (CTM) serves as a storage of so-called control token packets (CTP). There is a corresponding control token packet for every potentially executable instruction that has received at least one required control token. The structure of a control token packet is illustrated in Figure 2. Node address (NA) indicates the address of the instruction, also called as node packet, corresponding to this CTP. By means of the color indicator (CI), simultaneous or successive invocations of same subroutine may be performed. Status data (SD) contains information about the state of CTP as enumerated below :

1. CTP is being used or not.
2. If being used, does it contain all the necessary control tokens already? In the former case it is said to be a CCTP, an acronym for complete control token packet.
3. Being a CCTP, is the corresponding instruction being executed or not?
4. In case a CCTP has been executed already but some of its output control tokens are not yet issued due to unsafeness, such CCTP is newly dubbed as result control token packet (RCTP).

LCT and RCT stands for left control token and right control token, respectively. These binary indicators are set to 1 if the corresponding control token is available or dummy otherwise, it is cleared. Therefore, the completeness of a CTP is simply determined by the logical AND of LCT and RCT. CTSIs, control token sent indicators, show that corresponding destinations have received the output control token already if it is set to 1 or else it contains 0.

CONTROL FLOW NODES

Program is written in a control flow language

which is a collection of nodes almost similar to those defined by Dennis (see Figure3). Control flow nodes and their firing rules are shown in Figure 3a. With the exception of a subroutine node, a node becomes executable if all the needed control tokens are available. The principle of by availability control mechanism seems to be not suitable in the case of a subroutine node because it is actually an aggregate of the previously stated fundamental nodes. Therefore, an arrival of at least one control token, perhaps, may result into fireability of some fundamental nodes.

We decided not to use the input arc selector node and merge node (see Figure 3b). An input arc selector node will be of importance only when the data corresponding to the contol tokens to be selected are both constants. This will be a very rare case and it can be easily shown that an input selector node may be realized by using a pair of T-Gate and F-Gate nodes. The merge node, on the other hand, serves no purpose but to increase the number of nodes used in writing a control flow program thus slowing down its execution. This is because, if a program is accurately written, there is no possibility that a multi-source control arc will have more than one control token at any time for it will result into program non-determinacy.

CONTROL FLOW INSTRUCTION FORMAT

Figure 4 illustrates the format of a control flow instruction or node packet. The first two words (at most) contain information about the operation code, operand and result specifications, number of destinations, and destination specification. If existent, a left operand may be an immediate data or it may be memory referenced. The latter is classified into absolute, indirect or relative addressing. The right operand specification classification is almost the same as that of the left operand. Nevertheless, when the right operand's address is the same as that of the left operand, it may be specified in that manner so as to minimize the number of memory words occupied by a node packet. Similarly, in case the memory referenced result is the same as that of either left operand or right operand it may be specified so.

The format of destination specification is shown in Figure 5a. If SNI (subroutine node indicator) is equal to 1, the remaining five bits are interpreted as depicted in Figure 5b. WCA signifies which control arc will be triggered. If ENT equals 1, that is, the subroutine node will be "entered", the control arc to be triggered is an input one or else it is an output one. The corresponding destination address will be 16-bit long indicating the invoked subroutine number or name. If SNI is 0, the first three remaining bits will be interpreted as in Figure 5c while the last two remaining bits will be unused. Destination addressing type is either absolute or relative

but as much as possible relative addressing will be utilized. If NCTN, number of control tokens needed, equals 0, the token packet of the indicated node address needs only one control token to become executable or else it requires two. This information will be used to determine whether the right control token is dummy or not.

CONTROL FLOW PROGRAM

The instruction set summary of assembly language to be used in control flow programming is shown in Figure 6. This is based upon that used for MC68000.

As an illustration, consider the simple control flow program graph of Figure 7a. Note that the first two nodes both require one control token only. This is allowable provided that during the arrival of the required control token of the trigger node, valid data have been stored already into addresses A and B. The body of the program as defined in control flow assembly language is shown in Figure 7b.

Figure 8a illustrates the general format of a statement. It consists of statement number or label, operation code, operands and result field, and destination field. Copy, synch and trigger nodes do not require operands and result field. The other exception is an OAS node whose destination field is subdivided into TRUE-destination and FALSE-destination. This format is shown in Figure 8b.

The subroutine definition, as shown in Figure 9, is similar to that of a conventional assembly language's macro definition. It consists of three parts : header, body and trailer.

SIMULATION RESULTS

Using the previously mentioned control flow assembly language, a 16-point Fast Fourier Transform patterned program is simulated. Figure 10 shows execution time as a function of the number of functional units. We can see that the execution time stabilizes, that is, there is no apparent improvement, near the point where the number of functional units equals 32. This is due to the fact that, ideally, 16 functional units may be active simultaneously and, 16 more units are required so that soon to be created 16 complete control token packets may be immediately executed. The execution time corresponding to the stability range may not be the minimum due to other factors such as memory contention. However, such discrepancy, if there is, is minimal.

Figure 11 is a plot of the number of needed control token memory words versus the number of functional units. In order to avoid deadlock due to shortage of CTM words, the actually needed number of CTM words must be twice the simulation results. This is because control

tokens needed for the execution of a certain node packet may be simultaneously created. Hence, they will be written into separate CTPs and then combined. It can be shown that for any program such deadlock will most unlikely occur if the number of CTM words is Dp * (No + 2). Dp is the maximum parallelism of the program and No is the maximum number of output control tokens issued by any node.

CONCLUSION

Parallel control flow computer, due to its non-importation of non-control flow concepts, may turn out to be a better parallel processing machine. Above all, some of the programming methodologies developed for Von Neumann computer may be adopted because the latter operates upon the principle of (sequential) control flow.

REFERENCES

[1] Ackerman, W. B., and J. B. Dennis, "VAL - A Value Oriented Algorithmic Language, Preliminary Ref. Manual," Tech. Rep. TR-218, Laboratory for Computer Science, MIT, June 1979.

[2] Arvind, K. P. Gostelow, and Wil Plouffe, "An Asynchronous Programming Language and Computing Machine," Tech. Rep. TR-114a (UC-Irvine, Calif.), December 1980.

[3] Dennis, J. B., and D. P. Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," IEEE Proc. on 2nd Annual Symp. on Computer Architecture (1975), pp. 126 - 132.

[4] Sowa, Masahiro, "Control Flow Parallel Computer Architecture," IPS of Japan, Sig. Archi., 48 - 2, March 1983.

[5] Sowa, M., and T. Murata, "A Data Flow Computer Architecture with Program and Token Memories," IEEE Trans. on Computers, September 1982, pp. 820 - 824.

[6] Treleaven, P. C., D. R. Brownbridge, and R. P. Hopkins, "Data-Driven and Demand-Driven Computer Architecture," Computing Surveys, March 1982.

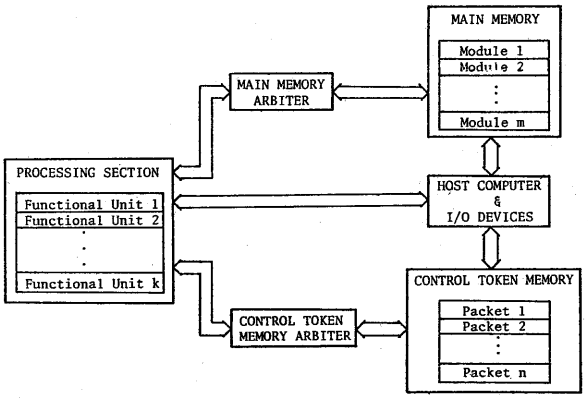


Figure 1. Control Flow Parallel Computer

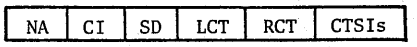
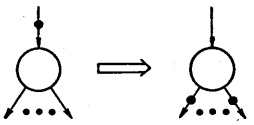
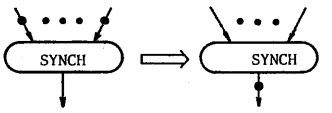


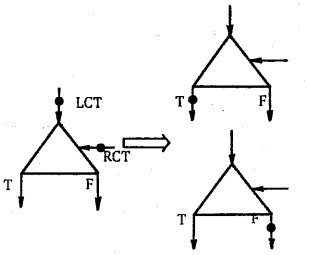
Figure 2. Control Token Packet



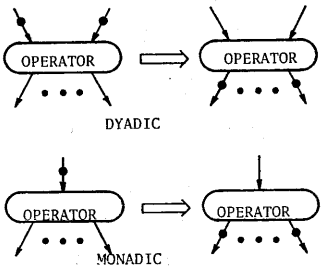
1. TRIGGER or COPY NODE



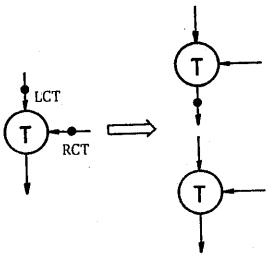
2. SYNCHRONIZE NODE



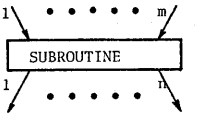
3. OUTPUT ARC SELECTOR (OAS) NODE



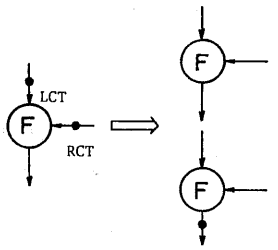
4. ARITHMETIC and BOOLEAN OPERATOR NODE



5. T-GATE NODE

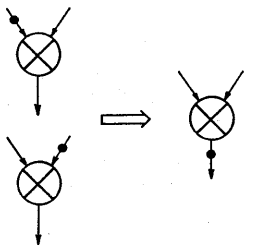


6. SUBROUTINE NODE

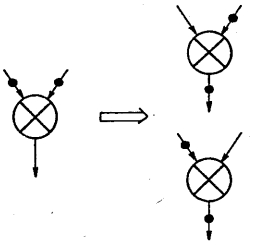


7. F-GATE NODE

a) Control Flow Nodes

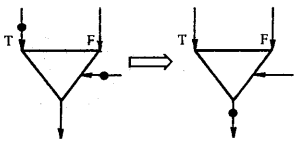


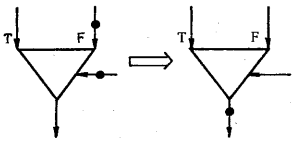
Determinate

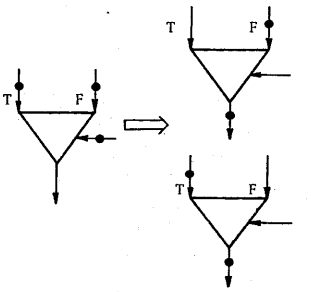


Non-Determinate

1. MERGE NODE







2. INPUT ARC SELECTOR NODE

b) NOT VERY USEFUL NODES

Figure 3. DENNIS Nodes (Actors)

OPCODE	LOS	ROS	RS	ND	DS1	DS2
DS3	DS4	DS5	DS6	DS7		
LEFT OPERAND						
RIGHT OPERAND						
RESULT						
DESTINATION 1						
DESTINATION 2						
.						
.						
DESTINATION 7						

Legend :

LOS - Left Operand Specification
ROS - Right Operand Specification
RS - Result Specification
ND - Number of Destinations
DSi - ith Destination Specification

Figure 4. Instruction Format

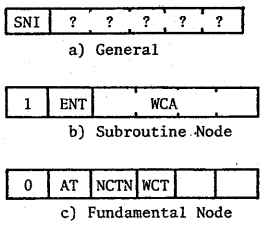
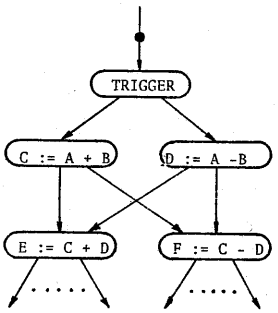


Figure 5. Destination Specification Format



a) Control Flow Graph

```
.
.
.
1   TRIGGER                    2,3
2   ADD        A,B,C           4,5
3   SUB        A,B,D           4,5
4   ADD        C,D,E           ?
5   SUB        C,D,F           ?
.
.
.
```

b) Assembly Language Program

Figure 7. Example Program

```
LABEL   OPERATION CODE   LEFT OPERAND,RIGHT OPERAND,RESULT   DEST. 1,DEST.2,...,DEST. n
```

a) General

```
LABEL     OAS        BOOLEAN-VALUED VARIABLE   TRUE-DEST./FALSE-DEST.
```

b) Output Arc Selector Node

Figure 8. Format of Control Flow Statement

```
NAME      SUBROUTINE      ARG1,ARG2,...,ARGn

          [BODY OF SUBROUTINE]

          ENDS
```

Figure 9. Subroutine Definition

Legend :

[EA] - effective address
1 byte - 16 bits
1 word - 2 bytes
1 long word - 2 words

I. Data Movement Operations

1. MOVE [EA],[EA]
   The size of the operand may be specified to be byte, word or long as illustrated subsequently: MOVE.B, MOVE.W, MOVE.L
   MOVEC [EA],[EA],[EA]
   The number of bytes to be moved is always 1 more than what is indicated in the third [EA] which is 1-byte long.
   MOVEI #[DATA],[EA]
   The size of the immediate data matches the operation size.
2. EXG [EA],[EA]
3. SWAP [EA],[EA]

II. Integer Arithmetic Operations

1. ADD [EA],[EA],[EA]
   ADDI [EA],#[DATA],[EA]
   ADDI #[DATA],[EA],[EA]
2. SUB [EA],[EA],[EA]
   SUBI [EA],#[DATA],[EA]
   SUBI #[DATA],[EA],[EA]
3. MULS [EA],[EA],[EA]
   MULU [EA],[EA],[EA]
4. DIVS [EA],[EA],[EA]
   DIVU [EA],[EA],[EA]
5. INCR1 [EA],[EA]
   INCR2 [EA],[EA]
   INCR4 [EA],[EA]
6. DECR1 [EA],[EA]
   DECR2 [EA],[EA]
   DECR4 [EA],[EA]
7. NEG [EA],[EA]

III. Multiprecision Arithmetic Operations

1. ADDR [EA],[EA],[EA]
   ADDRD [EA],[EA],[EA]
2. SUBR [EA],[EA],[EA]
   SUBRD [EA],[EA],[EA]
3. MULR [EA],[EA],[EA]
   MULRD [EA],[EA],[EA]
4. DIVR [EA],[EA],[EA]
   DIVRD [EA],[EA],[EA]

IV. Logical Operations

1. AND [EA],[EA],[EA]
   ANDI [EA],#[DATA],[EA]
   ANDI #[DATA],[EA],[EA]
2. OR [EA],[EA],[EA]
   ORI [EA],#[DATA],[EA]
   ORI #[DATA],[EA],[EA]
3. EOR [EA],[EA],[EA]
   EORI [EA],#[DATA],[EA]
   EORI #[DATA],[EA],[EA]
4. NOT [EA],[EA]

V. Shift and Rotate Operations

1. ASL [EA],[EA],[EA]
   ASR [EA],[EA],[EA]
   LSL [EA],[EA],[EA]
   LSR [EA],[EA],[EA]
   ROL [EA],[EA],[EA]
   ROR [EA],[EA],[EA]
2. ASLI [EA],#[DATA],[EA]
   ASRI [EA],#[DATA],[EA]
   LSLI [EA],#[DATA],[EA]
   LSRI [EA],#[DATA],[EA]
   ROLI [EA],#[DATA],[EA]
   RORI [EA],#[DATA],[EA]

VI. Compare Operations

1. EQ [EA],[EA],[EA]
   GE [EA],[EA],[EA]
   GT [EA],[EA],[EA]
   LE [EA],[EA],[EA]
   LT [EA],[EA],[EA]
2. EQI [EA],#[DATA],[EA]
   EQI #[DATA],[EA],[EA]
   GEI [EA],#[DATA],[EA]
   GEI #[DATA],[EA],[EA]
   LEI [EA],#[DATA],[EA]
   LEI #[DATA],[EA],[EA]
   LTI [EA],#[DATA],[EA]
   LTI #[DATA],[EA],[EA]

VII. Bit Operations

1. BTEST [EA],[EA],[EA]
   BTESTI #[DATA],[EA],[EA]
   BSET [EA],[EA],[EA]
   BSETI #[DATA],[EA],[EA]
   BCLR [EA],[EA],[EA]
   BCLRI #[DATA],[EA],[EA]

Figure 6. Instruction Set Summary

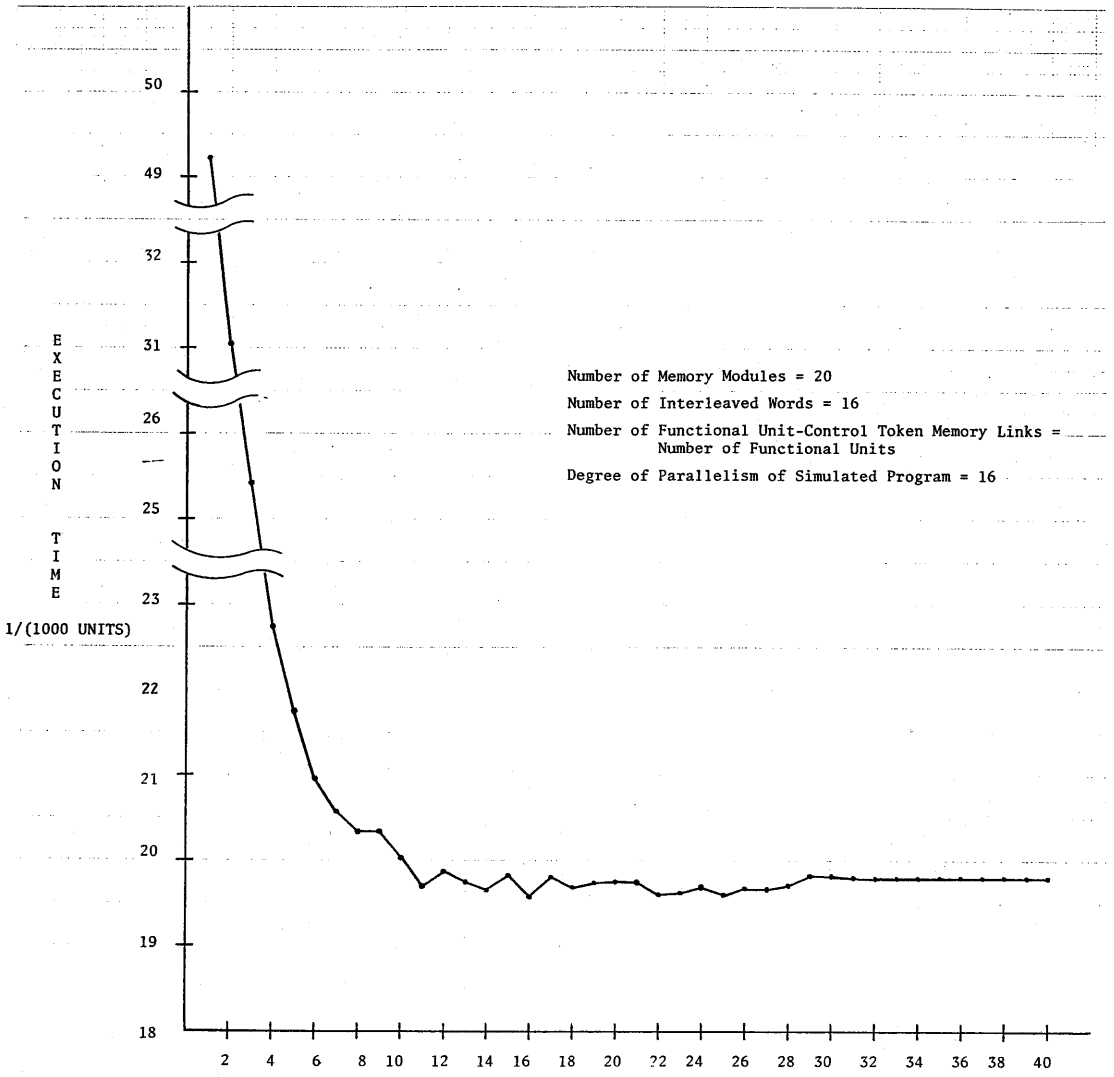


Figure 10

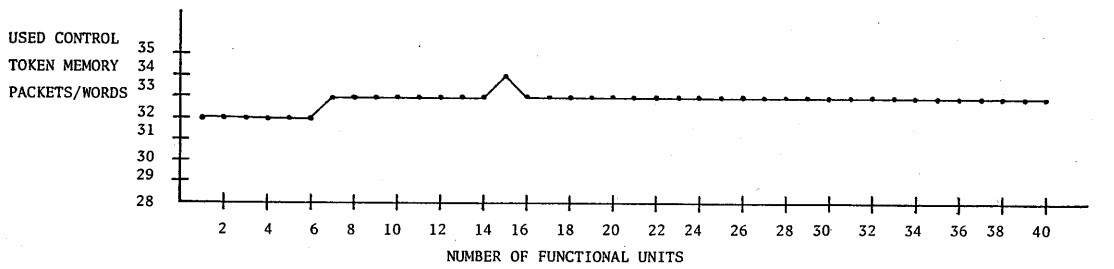


Figure 11

SIMULATION RESULTS