Research Report

KSTS/RR-84/006
24 Dec., 1984

Not (Operation): A Solution to Matching Bottleneck Problem in Dataflow Computing Models

by

Jayantha Herath, Nobuo Saito, Kenji Toda, Yoshinori Yamaguchi, Toshitsugu Yuba

J. Herath, and N. Saito
Department of Mathematics, Keio University
K. Toda, Y. Yamaguchi, and T. Yuba
Electrotechnical Laboratory

Department of Mathematics
Faculty of Science and Technology
Keio University

Hiyoshi 3-14-1, Kohoku-ku, Yokohama, 223 Japan

KSTS/RR-84/006
December 24, 1984

NOT(OPERATION): A SOLUTION TO MATCHING BOTTLENECK PROBLEM IN DATAFLOW COMPUTING MODELS

Jayantha HERATH, Nobuo SAITO*,*
Kenji TODA, Yoshinari YAMAGUCHI, Toshitsugi YUBA
Electrotechnical Laboratory,
1-1-4, Umesono, Sakuramura, Niiharigun,
Ibaraki, 305, Japan.
Department of Mathematics,
Keio University,
Yokohama, Japan.

Abstract

This paper first describes the importance of the dataflow computing machine architecture for new generation computing machines. The matching bottleneck problem of dataflow computing machine models will be introduced by giving actual experimental results obtained by simulation. The reasons for this situation are discussed, and an effective solution using a novel NOT(OPERATION) control mechanism is introduced. A new low level instruction set is defined from this. The effectiveness of the control mechnism is first evaluated at the definition level, then at the dataflow computing graph level, the very first stage of application. Here, the critical path length of the dataflow computing graph and number of arcs in a dataflow computing graph are used to show the effectiveness of the mechanism. Finally, the effectiveness is evaluated at the dataflow computing machine level by implementing the mechanism in the EM-3's virtual machine, a parallel, pipelined, dataflow computing scheme. Here, the performance characteristics measurements are obtained by executing benchmark programs using the software simulator. The results obtained are presented and discussed.

1. Introduction

Until now, computing has been dominated by von Neumann, or sequential mono processing. The existing von Neumann computing environment was introduced over 30 years ago. Hardware development alone resulted in four computing machine generations with this architecture. Although there are electronic computers operating with a cycle time of 10 ns, there have been many attempts to develop circuit and processor components which will operate still faster. Although the conventional processors can be speeded up just by increasing the speed of circuit components, there is a physical limitation to the speed, such as the propagation speed of signals in a physical media, that can be achieved using technology. Further, existing computing processors do not fully exploit existing technology. Technology is not the only way to achieve high performance computing environments. Parallel processing in computing systems offers an alternative to sequential processing. Parallel processing organizes problems quite differently from sequential processing. In principle, further parallel processing has no limitation on concurrent actions, hence it gives the maximum speed that can be obtained by parallel processing a given algorithm using a high speed computing system.

von Neumann architecture uses the basic concept of control logic which results in a global memory cell for storing sequentially executable instructions written in procedural languages such as Basic or Fortran, data, and a CPU to execute the instructions one by one. The one by one approach is the limiting factor of the machine performance. This is the von Neumann bottleneck that caused the software crisis which created the necessity for a new generation of computing machine environments.

1.1 Dataflow computing machine architecture

More radical departure from the von Neumann system is the dataflow system. In dataflow, an instruction is executed when and only when all the input operands are available, and this execution is performed immediately after the arrival of operands. The datflow concept abolishes central control in computing machines. Dataflow models execute instructions written in single assignment languages such as ID and EMLISP. Here, the variables are assigned only one value during a program's evaluation. The computations are free of side effects, and independent computations may proceed naturally in parallel. The dataflow concept exploits all the parallelism available in the algorithm, or exploits the parallelism inherent in the program at the architecture level. This is the most attractive feature of dataflow concept.

The dataflow approach has the potential of efficient large-scle exploitation of concurrency, maximum utilization of VLSI in computer design, compatibility with distributed networks, and compatibility with functional high-level programming. It is highly likely that dataflow computing machine architecture will be the next architecture generation of computing machines. Many research institutions in Japan, USA and Europe are making an effort to realize the practical dataflow machine. A variety of datflow architectures has been proposed, and hardware prototypes have been constructed from some of these proposals. Hardware prototypes must execute larger benchmark programs which can not be executed in a software simulator. This a necessary condition to be satisfied when constructing hardware prototype models. The performance characteristics measurements of some of these proposals have been evaluated by executing small scale dataflow flow programs using software simulators or hardware prototypes. It is necessary to prototype numerous ideas in various machine models, as this will help decide the most succesful dataflow architectural model, and will help solve problems related to the datflow computing system environment. The power of dataflow concept is better suited to special purpose applications than general purpose applications. The dataflow concept can be easily implemented either in numerical computations such as scientific and technological computations, or in non numerical computations such as symbolic manipulation in knowledge based information processing systems, because the machines can be designed and tuned for a specific purpose to gain speed and efficiency.

1.2 Outline

`pplication of the dataflow concept gives a powerful architectural model for computing. In the following sections, languages for dataflow computing models are first described. The matching section bottleneck identified in the dataflow architectural model and reasons for this problem, i.e. conditional operation implementation in dataflow computing graphs, are then described. An example illustrates the problem and its cause by giving experimental results. A solution to the problem is introduced using a new NOT(OPERATION) control mechanism. An example shows how to implement the control mechanism. The effectiveness of this mechanism is evaluated at the definition level, then at the very first stage of application, i.e., the level of dataflow computing graphs, and finally at the dataflow computing machine level by implementing the mechanism in the EM-3, a dataflow computing machine's architectural model. Finally, using the virtual virtual machine that we have constructed, the performance characteristics of the machine are measured, compared and discussed. Appendix shows the EM-3 dataflow architectural model.

2. Languages for dataflow computing machines

The medium of communicating with the computer system, the language, is very important in writing programs to implement parallel algorithms in the real dataflow computing machine. It is very easy to program in high level languages with any computer model. High level programming languages for dataflow machines is also a very interesting and important research field in which no positive research results have yet appeared. Functional programming languages do not reflect the properties of the von Neumann computing model. Side effect free functional languages such as Pure Lisp, based on Church's lambda calculus, and Backus's Fb)ased on Curry's combinatory logic can be used effectively to execute programs in datflow machines. Single assignment languages can be considered as functional languages. There is a lot more work to be done in this field to implement these languages. Functional semantics simplify the translation process. In functional programming, programs are put together with some operation to construct a new program which can be used to construct larger programs. However, it is necessary to design dataflow languages for dataflow computing machines with the dataflow concept in mind. In designing dataflow languages, programmers should not consider the explicit control of memory allocations, but should deal only with data values. The LISP programming language group developed over the last 25 years seems to be the most powerful language group for symbolic manipulations. The PROLOG language group developed over the last 13 years in Europe also has the power to implement logic programming in dataflow machines.

Low level languages for dataflow computing machines should describe dataflow graphs efficiently.

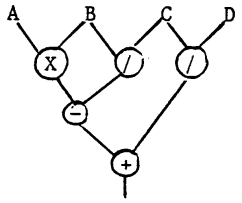


Fig. 1(a). Dataflow computing graph AxB - B/C + C/D

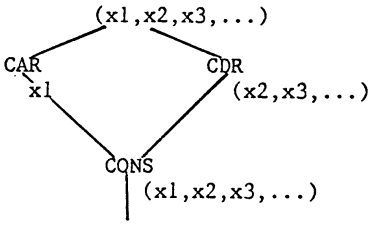


Fig. 1(b). Dataflow computing graph CONS(CAR(x1,x2,x3..),CDR(x1,x2,x3..)).

Fig. 1 Dataflow graphs - numerical and non numerical

Fig. 1(a) shows the dataflow graph for the numerical computation

A * B - B / C + C / D

Fig. 1(b) shows the dataflow graph for computing the non numerical computation

CONS(CAR(x1,x2,....), CDR(x1,x2,....)).

The high level programs in this paper are described in EMLISP, a single assignment language specially designed for dataflow computing. EMLISP semantics support functional programming. In EMLISP all iteration calculations are supported by recursion, but no global variables are supported.

EMIL is the low level language used to describe the dataflow computing graphs. Representation of dataflow graphs using the EMIL low level language will be illustrated in the following sections.

3. Matching bottleneck problem

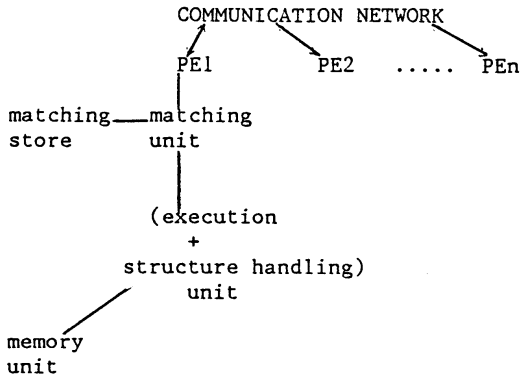


Fig. 2 Basic dataflow computing machine organization

Fig. 2 shows a very basic dataflow computer organization. Processing elements are connected via a communication network. A processing element basically consists of a matching unit, execution and structure handling unit, and a memory unit. The communication network is used to interconnect the processing elements. The execution and structure handling unit performs the execution and structure handling in dataflow computing.

Operations can have either one or two operands. If there is only one operand needed to execute an operation it is easy to execute that operation immediately. When the operation is a two operand operation, the operation has to wait until both the operands are available, and the operation must receive the exact two operands whose origins are different. There must be an operand matching section in every dataflow machine model. The matching section is used to match the plural operands directed to the same operation. Operands wait in the matching section until their partners arrive. The matching section sequentially matches and synchronizes the operands. Matched partners are sent to be executed. The larger the number of two operand operations to be executed in a program, the more time spent in the matching section. This delays the execution of the whole program. Therefore, this is the most critical unit. This condition is called the matching bottleneck in dataflow computing machine architecture. This is very similar to the von Neumann bottleneck in conventional computers.

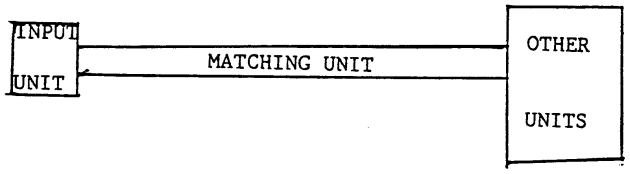


Fig. 3 Matching bottleneck

One of the problems to be solved in dataflow architectural models, is to eliminate this very narrow matching bottleneck, shown in Fig. 3, in the input to the matching section where the matching of the two operand operations were performed. This seems to be one of the important dataflow computing machine architectural problems to be solved, since this matching bottleneck is the limiting factor of the performance of all types of dataflow models. This problem is unavoidable in every dataflow machine using a matching section to match operand partners of an operation. Possible causes of this situation were carefully investigated, and

it was observed that the conditional operation implementation in dataflow graphs creates the matching bottleneck. This is described in the following section.

4. Conditional operation implementation in dataflow machines

Conditional expressions are the most important expressions in programming, and are used to compute a value based on some conditional predicate. The conditional operation interpretation used in parallel computing systems is the same as conventional computing systems, particularly in dataflow computing systems. First, we will consider the conditional operation implementation in dataflow computing. Consider the following conditional expression.

IF C(x) = TRUE
THEN B1(x), B2(x), ...
ELSE A1(x), A2(x),.... .

This is the basic conditional operation definition used for von Neumann computing. There is no basic change in implementing conditional operations in parallel systems except that the number of executions, for Ex. here B1(x), B2(x),... or A1(x), A2(x),.... , are allowed to process in parallel. The implementation in dataflow computing uses special SWITCH operations. Fig. 4 shows how the above conditional expression is implemented in dataflow computing.

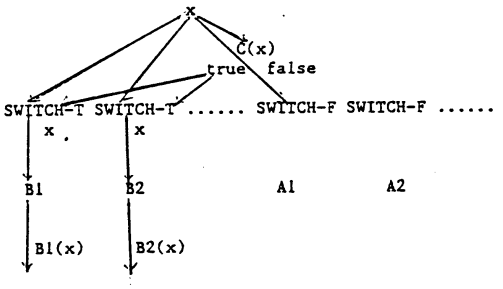


Fig. 4a. Conditional expression IF C(x) THEN B1(x), B2(x)... Implementation

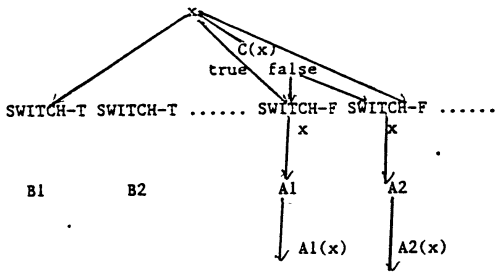


Fig. 4a. Conditional expression ELSE A1(x), A2(x)... Implementation

Fig. 4 Conditional expression IF THEN ELSE implementation

Fig. 4(a) 'C' is any condition operation. If 'x', the data value input to the condition operation, satisfies the condition 'C', the output of the operation is the TRUE boolean value. Otherwise, the FALSE boolean value is the output. Boolean values are carried by control arcs. The values of the type integer, real, or string are carried by the data arcs. Either of these output boolean values controls the number of SWITCH-T and SWITCH-F operations in the following step. If the control input to the SWITCH operations, the output of the condition operation, is TRUE , only SWITCH-T operations switch the 'x' data value to the following step. SWITCH-F operations do not give any output. This is illustrated in Fig. 4(a). Here, IF C(x) = TRUE

THEN B1(x), B2(x), ... is executed. C(x) is TRUE and therefore the SWITCH-T operations switch the data value 'x' to the operations B1, B2, ... to execute, and these operations give B1(x), B2(x),... data value outputs.

If the control input of SWITCH operations is FALSE, only SWITCH-F operations switch the data value input to the output. No SWITCH-T operation give an output. Fig. 4(b) shows the execution of ELSE A1(x), A2(x) .... . Here, C(x) is FALSE, and therefore the SWITCH-F operations switch the data value 'x' to the operations A1, A2 .... to execute and these give the output A1(x), A2(x) ..... .

All SWITCH operations used are two operand operations. The purpose of using these operations is to copy the data value from the top input arc to output arcs. The selection is determined by the boolean output the of condition operation. This results in two operand operations for every single conditional operation, which may be a one operand or two operand operation. Therefore, conditional operation interpretations used in dataflow computing models contribute greatly in increasing the number of two operand operations to be execute, and hence to be matched. This increase in two operand operations is made artificially, and is the main reason for the matching bottleneck, and consequently high execution times for program execution in dataflow computing.

The conditional operation implementation described above always gives a boolean output which is not always necessary in dataflow computing. If the data value satisfies a particular condition, then that data will undergo a defined operation. Boolean values do not have to be generated. The implementation increases parallelism in the computation, increases the number of matching operations to be performed, and increases the number of operations to be executed unnecessarily. All these contribute to weaken the effectiveness of he dataflow computing concept. Unnecessarily large two operand operations cause the bottleneck in the matching section which limits the performance of the dataflow processor.

Switch operations are used for sequencing the flow of data in condition operations by using a boolean control. This adds the control driven property to dataflow computing. The computing must not generate unnecessary data values. The SWITCH operations degrade the higher performance obtainable by implementing dataflow computing. When more switch operations are involved in execution, the effectiveness of parallel processing in the dataflow scheme is diminished drastically. The number of switch operations to be performed in a practical program will be very high, and the resulting matching bottleneck will create very serious problems in realizing practical dataflow computing machines. It is necessary to build dataflow computing machines which have a much higher speed than conventional machines.

The example in the following section shows the real situation in implementing SWITCH operatons. The primitive operations used in the EM-3 dataflow computing machine, a sample program written in high level dataflow language, and its dataflow computing graph will be shown. The primitive operation's execution frequency when a program is executed will also be presented.

Example 1

Appendix 1 describes the EM-3 dataflow computing maching model under construction in the Electrotechnical Laboratory, Japan. We have constructed a virtual machine, a software simulator, to verify the dataflow computing principle and identify implementation problems at the software level. Small size benchmark programs are executed in this virtual machine to evaluate the performance of the EM-3 engineering prototype. The stream data structure concept, sequences of values with operations car, cdr, cons and null are used.

EMLISP is our high level dataflow computing programming language. EMLISP can be considered as a single assignment language, i.e. an object or a variable, designed with a value for it, may be assigned a value at most once during the program execution, implying the maximum inherent parallelism in a program. EMLISP is a member of the powerful LISP language group. PROG feature is a side effect facility used in LISP to increase the efficiency of sequential execution of programs in a von Neumann computing environment. Such features are removed from LISP to obtain side effect free, pure functional list processing in a highly parallel dataflow computing environment. EMLISP does not include relatives of PROG such as PROG, PROG2, PROGn, operations to control the flow such as GO and DO, modifying lists such as RPLCA, RPLCD, NCONC .... operations, or relatives of array such as ARRAY, STORE of the conventional LISP, but consists of special features such as parallel COND, parallel OR, parallel AND, and BLOCK for blocking. Block adds procedural programming style to EMLISP. Bound variables are defined inside the block, and S-expressions can be arraged in any order. Variable dependencies define the S-expression linkages in a block. PCOND adds the guarded command feature. Here, the propositional expressions are evaluated concurrently, and then if conditional are true the corresponding forms are evaluated. The programs were written in the dataflow language EMLISP. EMLISP design strategies are summerized below. Here global and free variables, loops and functions replacing the existing list structures are inhibited. Single assignment rule and pure functionality are adopted.

The low level dataflow LISP language used in EM-3 to construct dataflow graphs is EMIL. The format of EMIL operations is shown in Table 1.

```
(OPCODE CONSTANT DEST-LIST)
(CALL FUNCTION-NAME NO-OF-ARG NO-OF-RET DEST-LIST)
(PROC FUNCTION-NAME NO-OF-ARG DEST-LIST)

where  CONSTANT ::= (C-0 n) | (C-1 n)
       DEST-LIST ::= (DESTINATION)
       DESTINATION ::= (LABEL [ NIL | PORT-NO | {CONTROL | DATA} | (ARG u v)
                                                  |RETURN NO-OF RET) ] )


LABEL ::= string .
n     ::= integer
u     ::= integer
v     ::= integer
PORT-NO ::= integer
NO-OF-RET ::= integer
```

Table 1 Format of emil operations

A node in the dataflow graph is represented by either an operation (opcode) or a function name. Destinations of the result of an operation are described by dest-list and corresponds to the number of output arcs of a node in a dataflow computing graph. Constant type operands of an operation are placed in the constant datum field of the operation. A destination field consists of a label field and an attribute field. The label field represents the destination node. The attribute field represents the node attribute. Here, port-no represents the port number of the operation where the operand is destined, and argument 'u' in total 'v' arguments of the function called is represented by (arg u v). 'proc' specifies the function name, total number of arguments, and destination list of each argument.

```
Fib n = 1; if n=0 or 1
      = Fib n-1 + Fib n-2

(defun fibonacci (n)
       (cond (( zerop n) 1)
             (( eq n 1) 1)
             (t (plus (fibonacci (difference n 1))
                      (fibonacci (difference n 2))))))
```

Fig. 5(a). Fibonacci function and its EMLISP Program

```
(PROCEDURE FIB 1. (G0002 MONO-0) )
G0002 (*DISTRIBUTE (G0003 MONO-0) (G0006 DATA) (G0014 DATA) )
G0003 (*EQ (C-1 1.) (G0004 MONO-CONTROL) (G0007 MONO-CONTROL) )
G0004 (*SWITCH-T (C-0 1.) (G0018 (RETURN 1.)) )
G0006 (*SWITCH-F (G0005 0.) )
G0007 (*SWITCH-F (C-0 2.) (G0005 0.) )
G0005 (*EQ (G0008 MONO-CONTROL) (G0010 CONTROL) (G0011 MONO-CONTROL)
      (G0014 CONTROL) (G0015 MONO-CONTROL) )
G0008 (*SWITCH-T (C-0 1.) (G0018 (RETURN 1.)) )
G0010 (*SWITCH-F (G0009 0.) )
G0011 (*SWITCH-F (C-0 1.) (G0009 1.)) )
G0009 (*DIFFERENCE (G0012 (ARG 1. 1.)) )
G0012 (*CALL FIB 1.1. (G0017 0.) )
G0014 (*SWITCH-F (G0013 0.) )
G0015 (*SWITCH-F (C-0 2.) (G0013 1.) )
G0013 (*DIFFERENCE (G0016 (ARG 1. 1.)) )
G0016 (*CALL FIB 1.1. (G0017 1.) )
G0017 (*PLUS (G0018 (RETURN 1.)) )
G0018 (*RETURN 1.)
END
```

Fig. 5(b). EMIL code - Fibonacci

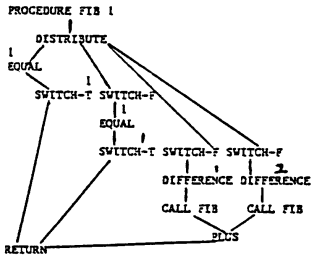


Fig. 5(c) Dataflow graph - Fibonacci

Fig. 5 Emil code and dataflow graph fibonacci

Fig. 5(a) shows an example of this high level programing language to compute the Fibonacci numbers. The Fibonacci function has binary tree parallelism and contains only numerical computations.

The functionality of the primitive, EMLISP operations may be summarized as:

List operations

car(()) = error
car(x1,x2,...) = x1
cdr(()) = error
cdr(x1,x2,x3...) = x2,x3,...
cons(x1 (x2,x3,....) = x1,x2,x3,....
equal((x x)) = true
equal((x y)) = false

Attribute checking

null(()) = true
null(x1...) = false
atom((a)) = true
atom(x1,x2..) = false
numberp((1...)) = false
numberp((1)) = true

IF x ¿ y

ssthan(x y) = false
lessthan(y x) = true
greaterthan(x y) = true
greaterthan(y x) = false

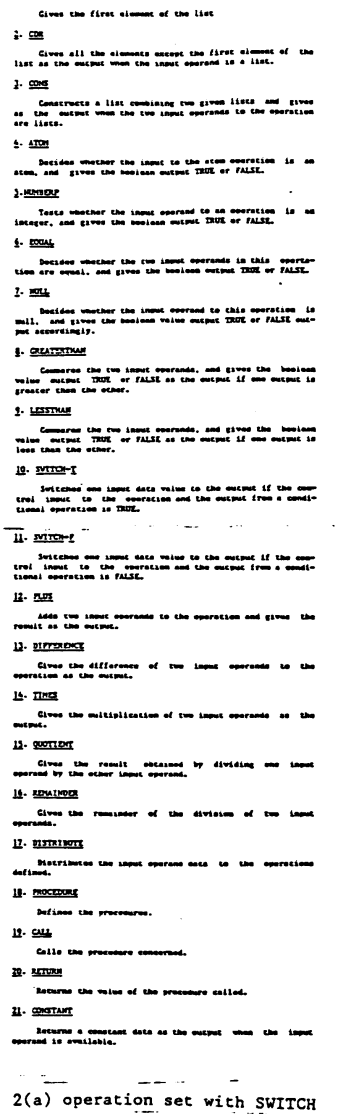
Numerical operations

plus (x,(y)) = x+y
difference(x,(y)) = x-y
times(x,(y)) = xy
quotient(x,(y)) = x/y
remainder(x,(y)) = rem x/y

Dataflow graph application codes

distribute
switch-t
switch-f
procedure
call
return

The EMIL code of the Fibonacci function is shown in Fig. 5(b), and the dataflow graph is given in Fig. 5(c).



Gives the first element of the list

2. CDR

Gives all the elements except the first element of the list as the output when the input operand is a list.

3. CONS

Constructs a list combining two given lists and gives as the output when the two input operands to the operation are lists.

4. ATOM

Decides whether the input to the atom operation is an atom, and gives the boolean output TRUE or FALSE.

5.NUMBERP

Tests whether the input operand to an operation is an integer, and gives the boolean output TRUE or FALSE.

6. EQUAL

Decides whether the two input operands in this operation are equal, and gives the boolean output TRUE or FALSE.

7. NULL

Decides whether the input operand to this operation is null, and gives the boolean value output TRUE or FALSE output accordingly.

8. GREATERTHAN

Compares the two input operands, and gives the boolean value output TRUE or FALSE as the output if one output is greater than the other.

9. LESSTHAN

Compares the two input operands, and gives the boolean value output TRUE or FALSE as the output if one output is less than the other.

10. SWITCH-T

Switches one input data value to the output if the control input to the operation and the output from a conditional operation is TRUE.

11. SWITCH-F

Switches one input data value to the output if the control input to the operation and the output from a conditional operation is FALSE.

12. PLUS

Adds two input operands to the operation and gives the result as the output.

13. DIFFERENCE

Gives the difference of two input operands to the operation as the output.

14. TIMES

Gives the multiplication of two input operands as the output.

15. QUOTIENT

Gives the result obtained by dividing one input operand by the other input operand.

16. REMAINDER

Gives the remainder of the division of two input operands.

17. DISTRIBUTE

Distributes the input operand data to the operations defined.

18. PROCEDURE

Defines the procedures.

19. CALL

Calls the procedure concerned.

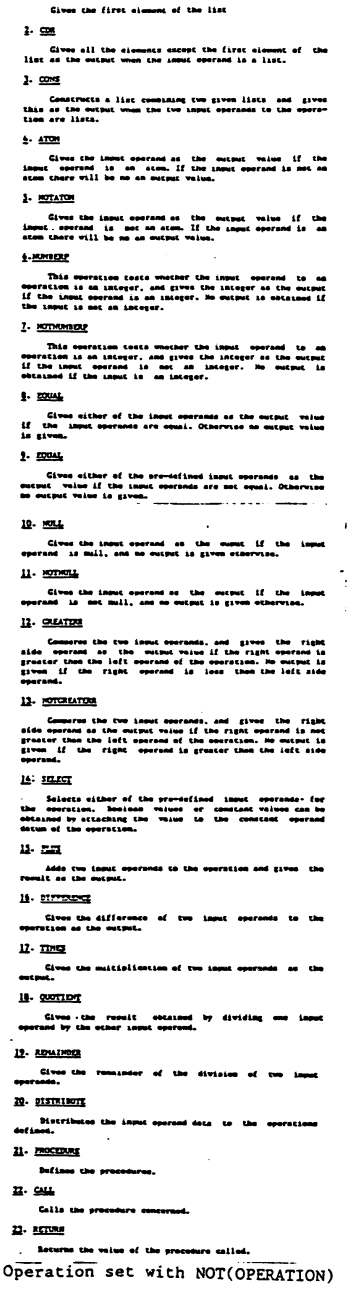
20. RETURN

Returns the value of the procedure called.

21. CONSTANT

Returns a constant data as the output when the input operand is available.

2(a) operation set with SWITCH



Gives the first element of the list

2. CDR

Gives all the elements except the first element of the list as the output when the input operand is a list.

3. CONS

Constructs a list combining two given lists and gives this as the output when the two input operands to the operation are lists.

4. ATOM

Gives the input operand as the output value if the input operand is an atom. If the input operand is not an atom there will be no an output value.

5. NOTATOM

Gives the input operand as the output value if the input operand is not an atom. If the input operand is an atom there will be no an output value.

6.NUMBERP

This operation tests whether the input operand to an operation is an integer, and gives the integer as the output if the input operand is an integer. No output is obtained if the input is not an integer.

7. NOTNUMBERP

This operation tests whether the input operand to an operation is an integer, and gives the integer as the output if the input operand is not an integer. No output is obtained if the input is an integer.

8. EQUAL

Gives either of the input operands as the output value if the input operands are equal. Otherwise no output value is given.

9. EQUAL

Gives either of the pre-defined input operands as the output value if the input operands are not equal. Otherwise no output value is given.

10. NULL

Gives the input operand as the ouput if the input operand is null, and no output is given otherwise.

11. NOTNULL

Gives the input operand as the output if the input operand is not null, and no output is given otherwise.

12. GREATER

Compares the two input operands, and gives the right side operand as the output value if the right operand is greater than the left operand of the operation. No output is given if the right operand is less than the left side operand.

13. NOTGREATER

Compares the two input operands, and gives the right side operand as the output value if the right operand is not greater than the left operand of the operation. No output is given if the right operand is greater than the left side operand.

14. SELECT

Selects either of the pre-defined input operands for the operation. Boolean values or constant values can be obtained by attaching the value to the constant operand datum of the operation.

15. PLUS

Adds two input operands to the operation and gives the result as the output.

16. DIFFERENCE

Gives the difference of two input operands to the operation as the output.

17. TIMES

Gives the multiplication of two input operands as the output.

18. QUOTIENT

Gives the result obtained by dividing one input operand by the other input operand.

19. REMAINDER

Gives the remainder of the division of two input operands.

20. DISTRIBUTE

Distributes the input operand data to the operations defined.

21. PROCEDURE

Defines the procedures.

22. CALL

Calls the procedure concerned.

23. RETURN

Returns the value of the procedure called.

2(b) Operation set with NOT(OPERATION)

Table 2 Basic definitions of EMIL operations

The EMIL code has the basic definitions shown in Table 2(a). NULL, ATOM, NUMBERP, EQUAL, GREATERTHAN and LESSTHAN are conditional operations and always give the boolean value output. NULL, NUMBERP, ATOM are single operand operations, and the other operations are double operand primitive condition operations. To sequence the flow of data in dataflow computing graphs, double operand

SWITCH-T and SWITCH-F operations are introduced and used after the primitive condition operations. This causes all condition operations to relate to double operand switch operations independent of the number of input operands in the condition operation. There are several versions of switch operations in use. In general, all these switches have the property of switching the input data to the output, depending on the boolean value output of the conditional operation which controls the switching operation. The number of switch operations to be added to sequence the flow of data is always greater than or equal to two. This makes the number of double operand operations to be matched in the matching section very high, making the matching section very busy, and creating a bottleneck there.

The bottleneck explained above is practically proved by our experimental dataflow computing system. We measured the performance characteristics of our system by executing several small scale benchmark programs. The n queen problem, quicksort algorithm, Fibonacci function, and some other non numerical and numerical problems are among the benchmark programs to be solved using the EM-3 dataflow system.

	TEST PROGRAM								
OPERATION	AK(2 9)	4QNS	4QNP	FIB(13)	QUI	SAF	COP1	COP3	AP
CALL	229	101	241	464	522	798	127		71
DISTRIBUTE	460	366	875	465	1383	1469	128	382	72
CONSTANT	0	3	3	0	0	0	0	0	0
SWITCH-F	1060	958	2488	2434	2246	2330	255	763	108
SWITCH-T	820	383	900	841	2115	1052	127	381	36
CAR	0	124	296	0	671	702	63	189	35
CDR	0	44	104	0	482	702	63	189	35
CONS	0	24	48	0	482	640	63	189	35
ATOM	0	0	0	0	0	126	127	381	0
NULL	0	39	90	0	483	0	0	0	36
EQL	350	162	390	841	0	736	0	0	0
PLUS	110	125	308	232	0	0	0	0	0
DIFFERENCE	229	44	100	464	0	0	0	0	0
TIMES	0	0	0	0	0	0	0	0	0
DIVIDE	0	0	0	0	0	0	0	0	0
PRINT	0	1	2	0	0	0	0	0	0
GREATERP	0	0	0	0	0	0	0	0	0
LESSP	0	0	0	0	189	0	0	0	0

Table 3 Execution frequency of operations

Table 3 shows the name of the operation, name of the benchmark program, and the corresponding frequency of primitive operation executed during the execution. These results show that the SWITCH and DISTRIBUTE operations comprise about 50This practical situation proves that the number of switch operations executed to sequence the flow of data when there is a condition is very high.

in the following section of this paper, a control mechanism to replace all the switch operations in dataflow computing graphs is presented and discussed.

5. Control mechanism to implement condition operations in parallel computing systems

A method to reduce the number of switch operations in the dataflow computing graph to minimise or eliminate entire switch operations and control arcs from dataflow computing is essential. A control mechanism with the ability to eliminate the incorrectly interpreted control driven feature in dataflow conceptual models is presented here and discussed in detail.

In all languages, it is necessry to delay the computation of either branch of a condition expression until the boolean value of condition operation is known. This boolean value gates the data values necessary for the execution of required operations.

A condition operation always consists of both positive and negative information about the operation. This can be viewed as the natural interpretation of condition operations. The mechanism introduced uncovers the fact that a condition operation always consists of a fundamental combination of positive and negative operations. This is used in this paper to eliminate the matching bottleneck. Here, any condition operation is represented as a combination of two basic operations. A condition 'C' is represented below by an 'OR' combinated 'C operation' and 'NOT C Operation'.

Condition 'C' = ('C' operation) OR ('NOT C operation')

DEFINITION A condition operation can be divided into two OR combinated positive and negative operations.

Since OR combinated operations can be executed concurrently in concurrent excution systems, both 'C' and 'NOT C' operations are executed concurrently to implement condition operations in a parallel dataflow computing environment. The particular significance of the NOT(OPERATION) in parallel systems, espeially in dataflow computing systems, is that these two joined operations can execute entirely independently, which increases the parallelism inherent in the program. This definition can be applied effectively to any parallel computing system. So far, a condition has been represented in a dataflow graph by adding switch operations, which resulted in two operand operations for any condition operation, and two sequential execution steps for the execution of conditional operations. The overall effect of the two operations which will replace any condition operation and produce a data value, is a more advanced , natural output than the boolean value output obtained from existing condition operations.

6. Control mechanism to implement condition operations in dataflow computing systems

NOT(OPERATION) with data value output

An extension of the NOT(OPERATION) introduced above will replace the control mechanism of condition operations efficiently, and will give completely new definitions to the condition operations in dataflow computing systems. This extension is described below.

In the basic definition of the conditional operation, two new, parallel operations can be used to generate boolean value outputs to control the concurrent switch operation set. The basic mechanism, NOT(OPERATION), uses negation successfully in condition operation execution. The extension generates data values instead of generating boolean values. The new operation can be considered as a successful .pplication of the negation technique in dataflow computing. The generation of boolean value output for every condition operation execution is abandoned. Boolean values can be obtained by executing some other operation when it is necessary. Either of the two operations which replace the condition operation produces a data output value and while the other produces no output value in the execution. This increases the real parallelism in the program execution, and simultaneously removes completely the unwanted pseudo parallelism inserted in the program to implement condition operations by introducing switch operations. This actually decreases the parallelism. Generally, the mechanism introduced here facilitates implementing condition operations in dataflow computing graphs by OR type parallelism.

Thus, the NOT(OPERATION) provides no delay in producing data values to be processed in the stage following a conditional operation. In the switch operation implementation, executing the condition operation results in a boolean value output. This initiates the execution of set of switch operations and this finally allows the execution of operations in the following stage.

The condition expression can be rewritten as the parallel execution of the following two expressions. Here, the IFNOT is used to replace the ELSE in defining conventional condition operations.

IF C(x) THEN B1(x), B2(x), ...

IF NOT C(x) THEN A1(x), A2(x), ...

The above definition is equivalent to the conditional expression used in section 4. This definition can be used in any BNF specification of syntax of a language. For example,

exp ::= .....IF expression THEN expression ELSE expression

recursion ::= IF expression THEN recursion ELSE recursion

can be rewritten as

exp ::= .....IF expression THEN expression

IF NOT expression THEN expression

recursion ::= IF expression THEN recursion

IF NOT expression THEN recursion

Fig. 6 illustrates the implementation of the above condition operation representation. The parallel representation of 'C' OPERATION and NOT'C' OPERATION replaces the whole configuration of Fig.4.

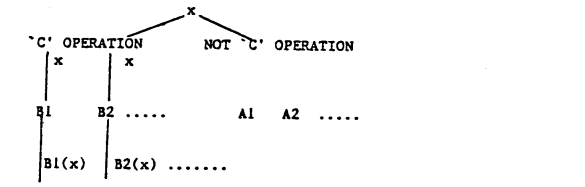


Fig. 6(a). Conditional expression IF C(x) THEN B1(x), B2(x)..... Implementation

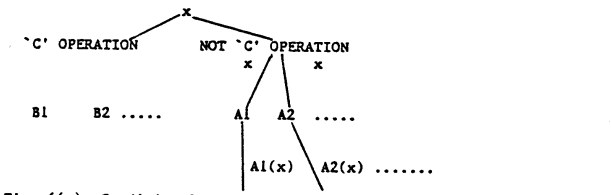


Fig. 6(a). Conditional expression IF NOT C(x) THEN A1(x), A2(x)..... Implementation

Fig. 6 Conditional expression implementation with NOTOPERATI

Fig. 6(a) shows the execution of IF C(x) THEN B1(x), B2(x), ... Here, the data value 'x' flows to both operations. If function C(x) is TRUE, then 'C' OPERATION will be executed, and the 'x' data value will be given as the output value and sent to the defined destinations. The execution of NOT'C' OPERATION will not give any output value in this instance. Fig. 6(b) shows the execution of the remaining part of the

conditional expression, i.e. IF NOT C(x) THEN A1(x), A2(x), ... Here the function C(x) is NOT TRUE, therefore the NOT'C' OPERATION will be executed, and the 'x' data value will be given as the output value to sent to the defined destinations. Here, the execution of 'C' OPERATION will not generate a output value. The interpretation clearly divides the operations to be performed after the condition 'C'.

7. Performance evaluation of NOT(OPERATION) method 1.

Definition level

The existing implementation of condition operations in dataflow computing machines always needs the help of more than two double operations in addition to the execution of condition operation to sequence the data. The execution takes place in two sequential steps. The new condition operation definition executes a conditional operation in a single step by using two operations. The parallelism in executing switch operation starts from 3/2, and increases according to the number of switch operations. This can be considered as unnecessary parallelism. The parallelism in new definition is always 2, can be considered as useful parallelism. Unless the condition operation to be executed is a double operand, no double operand operations are created to implement condition operations. The only operation created is another single operand operation which will execute simultaneously. If the condition operation is a double operand operation, one and only one extra double operation is created to implement the condition operations. Arcs drawn between operation nodes in a dataflow computing graph indicate sequencing the operations. Data in the form of communication packets are sent along the arcs to the instruction nodes. An instruction is executed only when all necessary data re present. The minimum number of arcs necessary to implement the single operand condition operation is seven in the existing implementation, and the minimum number of arcs necessary to implement the single operand condition operation with the new definition is four. The minimum number of arcs necessary to implement a double operand conditional operation with switches is eight, and the minimum number of operations is three. In new definition, the number of arcs is six, and number of operations is two. These are concurrent. The NOT(OPERATION) will give the low level dataflow machine languages the ability to express the conditional operations without using any form of SWITCH operations to sequence the flow of data.

Example 2 describes the implementation of this control mechanism. A new operation set is defined to substitute the operation set defined in Example 1. This will result in new dataflow computing graphs. The operation set is implemented in the dataflow computing model. The performance characteristics obtained are presented and compared with the performance characteristics obtained in Example 1, and discussed.

Example 2

To implement this control mechanism, it is necessary to redefine all the primitive operations used in low level dataflow computing languages. For example, if a machine uses LISP primitives to represent the low level language which describes the dataflow computing graph, it is necessary to redefine the primitive operations. A sequence of values with operations car, cdr, null, notnull, atom and not atom, is used in EM-3. The functionality of the operations may be summarized as

List operations

car(()) = error
car(x1,x2,x3....) = x1
cdr(()) = error
cdr(x1,x2,x3,....) = x2,x3,....
cons (x1 (x2,x3,x4,....)) = x1,x2,x3,....
equal(x x) = data output "(x)"
notequal(x x) = no data output
equal(x y) = no data output IF x =—= y
notequal(x y) = data output "(x)" IF x =—= y

Attribute checking

null (()) = data output "()"
null(x1..) = no data output
notnull(()) = no data output
notnull(x1,...) = data output "(x1,...)"
atom ((x)) = data output "(x)"
atom ((x1,x2,...)) = no data output
notatom((x1,x2,....)) = data output "(x1,x2,...)"
notatom(x) = no data output
numberp(1,...) = no data output
numberp(1) = data output "(1)"
notnumberp(1) = no data output
notnumberp(1,....) = data output "(1,....)"
greaterr(x, y) = data output "y" if y¿x
greaterr(x, y) = no data output if x¿y
notgreaterr(x, y) = no data output if y¿x
notgreaterr(x, y) = y if x¿y

Numerical operations

? x and y are integers
plus(x, y) = x + y
difference = x - y
quotient(x y) = x/y
remainder(x y) = rem of x/y

Dataflow graph application codes

distribute
procedure
call
return

The definitions used to implement the proposed mechanism are shown in Table 2(b).

ATOM and NOTATOM in the proposed mechanism have the ability to replace the ATOM operation and all the connected switch operations in the existing implementation.

EQUAL and NOTEQUAL replace the EQUAL operation and all connected switch operations involved in the operations followed by mentioned operations in the existing implementation.

The proposed NULL and NOTNULL operations replace the NULL and switch operations in the dataflow computing graphs.

The proposed NUMBERP and NOTNUMBERP operations replace NUMBERP and all related switch operations in the first version of dataflow computing graphs.

The proposed GREATERR and NOTGREATERR will replace GREATER THAN, LESS THAN and all other switch operations followed by these operations in dataflow computing.

We can implement GREATERL and NOT GREATERL operations which consider the left operands instead of considering the right side operands, in the proposed GREATERR and NOTGREATERR operations.

SELECT operations can be used to select the required input operand when necessary. This operation with constant, TRUE or FALSE boolean values attached to the input constant operand datum can be used to obtain the necessary value.

The proposed set of operations will give a complete operation set for the low level dataflow language based on LISP. It is possible to interpret the new feature added as the use of hierarchical modelling, to mean that the property of switch added basic condition operation . Up till now, conditional operation interpretations in realizing dataflow models were incorrect. The proposed control mechanism allows real dataflow in interpreting the dataflow concept when implementing practical dataflow machines. By correcting the

interpretation, we obtained a reduced instruction set. Every conditional expression is given a new definition. In Example 1, a minimum of three operations, the condition operation, SWITCH-T and SWITCH-F operations, have to be executed to allow the data to flow in the graph when executing conditional operations. This is done in two sequential steps. After implementing the NOT(OPERATION), the number of operations necessary to sequence the flow of data when executing conditional operations is two. Both these operations can be performed in parallel.

8. Performance evaluation of NOT(OPERATION) method 2.

dataflow graph application level

```
(PROCEDURE FIB 1. (G0002 MONO-0) )
G0002 (*DISTRIBUTE (G0003 MONO-0) (G0004 MONO-0) )
G0003 (*EQ (C-1 1.) (G0014 (RETURN 1.)) )
G0004 (*NEQ (C-1 1.) (G0006 MONO-0) (G00078 MONO-0) )
G0006 (*EQ (C-1 2.) (G0008 MONO-0)) )
G0007 (*NEQ (C-1 2.) (G0009 MONO-0) (G0010 MONO-0) )
G0008 (*CONSTANT (C-1 1.) (G0014 (RETURN 1.)) )
G0009 (*DIFFERENCE (C-1 1.) (G0011 (ARG 1. 1.)) )
G0010 (*DIFFERENCE (C-1 2.) (G0012 (ARG 1. 1.)) )
G0011 (*CALL FIB 1.1. (G0013 0.) )
G0012 (*CALL FIB 1.1. (G0013 1.) )
G0013 (*PLUS (G0014 (RETURN 1.)) )
G0014 (*RETURN 1.)
END
```

Fig. 7(a). EMIL code with NOT(OPERATION) - Fibonacci

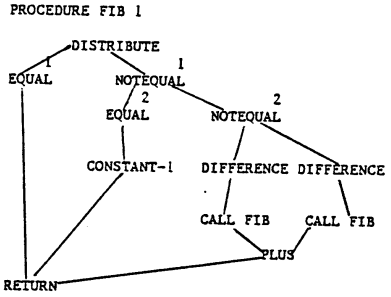


Fig. 7(b) Dataflow computing graph - NOT(OPERATION) Fibonacci

Fig. 7 EMIL code and dataflow computing graph with NOTOP-Fi

Fig. 7 shows the new version of the Fibonacci function dataflow computing graph. All the SWITCH-T and SWITCH-F operations are eliminated, and a parallel NOT EQUAL and EQUAL satisfies the necessary conditions. To calculate the critical path length of a dataflow computing graph distance betwen two sequential steps is counted as one unit. The critical path length measurement is taken to show the strength of the mechanism introduced, in the level of dataflow graph illustrations. The critical paths are shown in dark lines. The critical path length of the Fibonacci dataflow computing graph with switch operations is nine, and the critical path length of the NOT(OPERATION) Fibonacci dataflow computing graph is seven.

The arcs connecting the operations in a dataflow graph are related to the number of communication packets necessary to be generated. The total number of arcs is measured to show the strength of the mechanism in the early stage of application level.The total number of arcs in new version Fibonacci is ten, and in the old version it was fifteen. The reduction will result in lesser communication overhead. The number of communication packets used will be measured quantitatively in the execution of dataflow computing graph in the dataflow computing model, and is described in the following section.

9. Performance evaluation of NOT(OPERATION) Method 3.

Using EM-3 dataflow virtual machine.

The object of this simulation study is to observe the effect of the NOT(OPERATION) control mechanism in the dataflow computing environment. This can be done by constructing a software simulator, a virtual machine, of dataflow computer, and executing small benchmark programs on it. The software simulator used for this simulation study, simulation results, and the evaluation of the simulation results will be described in the following sections.

9.1 Software simulator

The software simulator describes the EM-3 dataflow computing machine and simulates the program behaviour faithfully. The simulator is written in SIMULA, therefore the parallel execution can be easily simulated using the "process" class object and timing parameters. The data structures, communication packets, processing elements of EM-3, each functional unit of a PE, and each store are expressed as class objects. Each functional unit has a packet queue in the input, and processes packet by packet, by taking the first element of the queue and despatching it to the output queue.

input to queue	1
output to queue	1
access time	
a store	1
result table	1
d-buffer	1
search time	1
router stage delay	1

	operation execution time	
operation	with SWITCH	with NOT(OPERATION)
CAR	2	2
CDR	2	2
CONS	4	4
ATOM	2	4
NULL	2	4
EQUAL	3	5
PLUS	3	3
DIFFERENCE	3	3
TIMES	7	7
PRINT	2	2
GREATERP	3	-
LESSP	3	-
SWITCH-T	1	-
SWITCH-F	1	-
NOTATOM	-	4
NOTNULL	-	4
NOTEQUAL	-	5
GREATER	-	5
NOTGREATER	-	5
CONSTANT	1	-
SELECT	-	3

Table 4 Timing parameters - software simulation

The simulation parameters used for operation execution are shown in Table 4. Compared to the earlier conditional operations, the value of the timing parameters used to measure the performance of NOT(OPERATION) has increased. Output packets directed to the same PE enter the input section with zero stage delay. The EMIL coded dataflow graph, i.e., the compiler output of the program written in EMLISP high level language, is the input to the simulator. The simulator interprets and executes a program written in EMIL code, the dataflow computing graph, and gives the performance characteristics of the EM-3. A binary router is used in simulating packet communication between PEs. The performance of the EM-3 is evaluated using this simulator [].

The effect of the NOT(OPERATION) introduced above is observed by rewriting the benchmark programs and executing them using the software simulator. The software simulator is constructed to give the effect of the new control mechanism. This is done by defining a new operation set. The performance characteristics of the machine are measured by executing these dataflow graphs in the simulator. The concurrency in Fibonacci(13) is shown in Fig. 8.

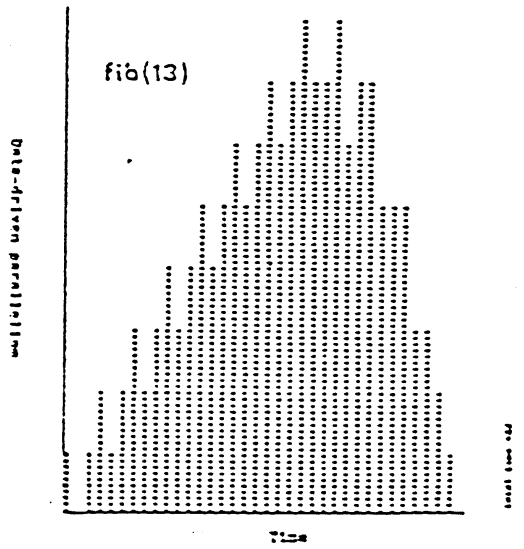


Fig. 8 Ideal datafow parallelism with switch operations

9.2 Evaluation of simulation results

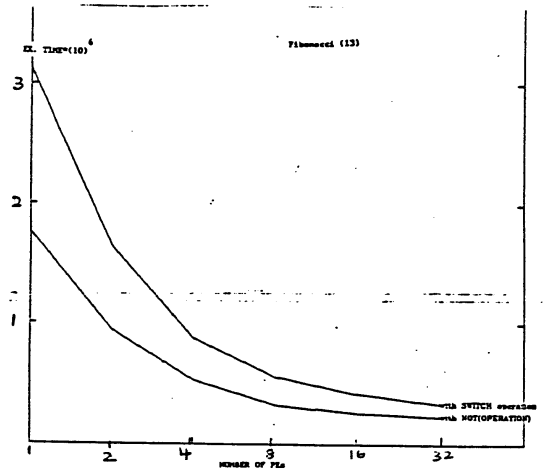


Fig. 9 Execution time with number of pe

The execution time of the benchmark program is measured by varying the number of processing elements. The results obtained for processing time with switch operations are plotted in graph (b) of Fig. 9. The results obtained for processing time with NOT(OPERATION) are plotted in graph (a) of Fig. 9. The shape of the graph does not change, but the execution speed approximately doubles.

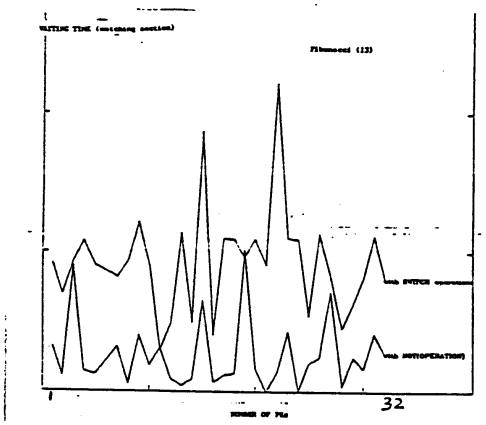


Fig. 10 Waiting time in matching section with pe

Fig. 10 shows the average waiting time variation in the matching section with a varied number of processing elements for programs written with switch operations and NOT(OPERATION). The waiting time is reduced to a comparably lower value when the NOT(OPERATION) is used. The packet waiting time in the matching section is reduced to one tenth of the earlier time.

Fig. 11 shows the maximum number of packets waiting in the queue to the matching section for a varied number of processing elements. The time is very low for the NOT(OPERATION).

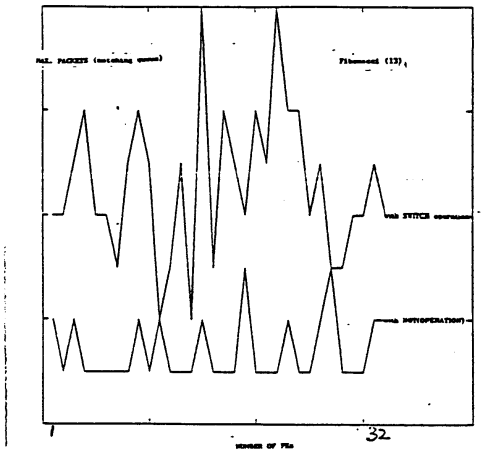


Fig. 11 Maximum number of packets in matching queue

Communication packets transfer the information between processing elements and sections inside a processing element. The packets travel to a defined destination node input, execute the operation with some other packets, and create another kind of communication packet. A communication packet created in a sub processing unit of a processing element carries the information necessary to be processed by some other sub-processing unit. The number of communication packets used in the program execution contributes directly to the communication overhead.

Result packets are used as external packets. One-operand packets are used to communicate with one-operand operations, and two-operand packets are used to communicate with two-operand operations which require matching to perform the function.

	notoperation	with switch
One operand packets	4148	8673
Two operand (Match) packets	464	4756

Table 5 Number of packets generated

Table 5 compares the one-operand packets and two-operand p ackets generated in the execution of Fibonacci(13). For example, the number of result packets generated to calculate Fibonacci(13) decreases from 8673 to 4148. The number of result packets entering matching section which were directly related to the two operand operations for Fibonacci(13) decreased from 4756 to 464.

Operation name	notoperation	with switch
distribute	465	465
constant	144	-
switch-t	-	841
switch-f	-	2434
notequal	841	-
equal	841	841
plus	232	232
difference	464	464

Table 6 Number of operations executed

Table 6 compares the number of operations executed. This shows the reduction in the number of operations executed.

section	not operation	with switch
Input	1	1
match	1	355
ifetch	3	4
preexec	2	2
exec	218	10
invo	1	1
exit	1	1
init	1	1
search	1	1
schedule	73	19
output	9	6

Table 7 Maximum number of packets in the queue with 1 PE

Table 7 shows the reduction in the maximum number of packets waiting in the matching section queue measured when the number of processing elements is one. Decrement of packets to be communicated implies the reduction of communication cost.

10. Conclusions

The candidates for the new generation computing machines will be

1. Highly advanced, parallel version of von Neumann computer architecture.
2. Non von Neumann computers with the ability to surpass the performance of conventional computing machines.
3. Combinations of von Neumann and non von Neumann machines.

This paper discussed the importance of dataflow computing concept which has the ability to challenge the conventional computing machines, and seems to be the most effective, promising computer machine architecture for new generation machines. The features of the engineering prototype of ETL dataflow computing machine, EM-3, now under construction, were discussed briefly, and some performance characteristic measurements were presented. The performance limiting factor of dataflow, the matching bottleneck, was introduced with practical measurements. The cause for this bottleneck is taken as the number of switch operations involved in executing dataflow programs.

The inefficiency of switch operations used in dataflow computing models was discussed, and an effective control mechanism was introduced to eliminate all the switch operations and which eliminated the matching bottleneck. A new instruction set must be defined to apply this control mechanism. This paper presented a new interpretation for condition operations by introducing a novel NOT(OPERATION). The new interpretation can be summarized as

condition 'C' = (C operation) OR (not C operation)

This interpretation can be considered as the very basic interpretation of condition operations. Negation is used successfully in conditional operation implementation in parallel computing systems. The idea is extended to support dataflow computing scheme by giving the data value output as the output of condition operation instead of giving the boolean value output.

The performance of the new mechanism was evaluated at the definition level. It was proved that any condition operation can be implemented in dataflow computing systems with the minimum of two parallel operations instead of implementing more than three operations executing in two sequential steps.

The performance of the mechanism was then evaluated at the level of application. The critical path length of a dataflow computing graphs were measured and compared, and the number of arcs in a dataflow computing graphs were counted and compared. This also proved that the new mechanism is effective.

Computing is performed in the dataflow environment thereafter to evaluate the performance of the mechanism. The proposed NOT(OPERATION) was implemented in the EM-3 software simulator and executed benchmark programs. The measured performance characteristics of the EM-3 were compared with switch operation implemented EM-3 dataflow model. The measurement comparisons included the execution times of benchmark programs, waiting times of two operand operations in matching section in the execution, frequency of operations executed in the execution, and the number of communication packets generated in the execution. The implementation of NOT(OPERATION) alone doubled the speed of the dataflow computing, the waiting time of the double operand operations in the matching section decreased to one tenth, and the communication overhead decreased. The new mechanism eliminated the switch operations completely, and reduced the number of sequential steps and number of operations to be executed. These factors contributed to eliminating the matching bottleneck and hence increased the speed of the machine. The mechanism will improve the efficiency of executing non deterministic programs in any dataflow computing machine model which currently employ switch operations. This mechanism automatically reduces remaining packet garbage collected in matching store and this is discussed in []. The application of NOT(OPERATION) to simplify logic program execution in the dataflow computing environment will be discussed in [].

Appendix

EM-3

The processing of the symbols standing for mental concepts, as opposed to numerical processing, is necessary in knowledge processing systems. The Electrotechnical Laboratory (ETL) of MITI, Japan, has been constructing a Lisp based dataflow machine for non numerical computations such as symbolic manipulations. This is the third generation of Lisp machines in ETL. The primary goal of the EM-3 project is to study the feasibility of the practical dataflow computing machine model for symbol manipulations. The EM-3, a superpersonal dataflow Lisp machine, aims at bringing out the intrinsic parallelism inherent in ordinary programs.

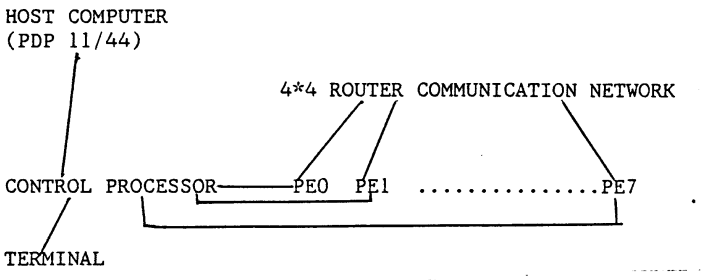


Fig. 1 Block diagram EM 3 prototype

The hardware prototype of EM-3 which we are constructing now, shown in Fig. 1. consists of eight identical processing elements connected via a 4 x 4 router network, which is a specially developed LSI chip.

In the EM-3 design, we are implementing new parallel processing control mechanisms such as pseudo-result, semi-result and partial-result in the dataflow computing environment. It has been proved that these notions revealed a new class of parallelism which was hidden in the execution in LISP group language instructions, and accelerated program execution in a parallel computing environment. A PE is a special hardware added MC68000 microprocessor. All the PEs in the EM-3 process communication packets in parallel.

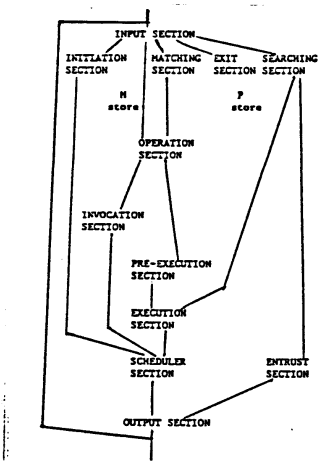


Fig. 2 Functional oraganization of a pe

The functional organization of a PE is shown in Fig. 2, and functions of each unit are described in []. The processing in each section is pipeline in theoritically but practically functions sequentially since the prototype is constructed using microprocessors. Each section of a PE takes in the first member of its input queue, processes it, and sends it to the output queue. Each section acts independently and asynchronously. A

detailed description of the prototype, functions of each section of a PE and the advanced control mechanism added to the dataflow computing are given in [],[] and []. The machine will be in full operation by the end of this year, and we are planning to construct the more advanced version of the EM-3, the EM-4, in which the reduced instruction set concept (RISC) is added to dataflow computing.

Acknowledgements

We are grateful to Professor Masahiro Ishii and Assistant professor Itoh Shuichi of the Applied Electronics department of the University of Electrocommunications, Tokyo, Japan, and Dr. Hiroshi Kashiwagi (Chief of the computer systems division, Electrotechnical Laboratory) for giving us the opportunity to carry out this research, and to the staff of the computer architecture section of the ETL for helpful discussions.

References

[1] Jack B. Dennis, "Dataflow supercomputers," *IEEE computer* (1980 Nov.), 48-56.

[2] D. D. Gajvaski, D. A. Padua, D. J. Kuck and R. H. Kuhn. "A second opinion on dataflow machines and languages," *IEEE Computer* (1982 Feb), 58-69.

[3] John Gurd and Ian Watson, "Preliminary Evaluation of a prototype dataflow computer," *IFIP* (1983), 545-551.

[4] John Gurd and Ian Watson," Datadriven system for high speed parallel computing part 1: Structuring software for parallel execution," *Electronic design* (1980 June), Vol.19, No.6, 91-100.

[5] Yamaguchi, Y., T. Yuba, T. Shimada; "A function evaluation mechanism of EM-3", *Papers of tech. Group on Comp., EC81-57, IECE Japan*, 1-8(Dec. 1981), In Japanese.

[6] Yuba T., Y. Yamaguchi, T.Shimada; "Lisp and a intermediate language for a dataflow machine," *Proc. 24th Nat. conf. IPS Japan*, 7D-7 (Mar. 1982), In Japanese.

[7] Yuba T., Y. Yamaguchi, T.Shimada; "A control mechanism of a Lisp based data-driven machine (EM-3)," *Inf. Proc. Lett.*, Vol.16, No.3, 139-143 (1983).

[8] Yoshinari Yamaguchi, Kenji Toda, Toshitsugu Yuba; "A performance evaluation of a Lisp based data-driven machine (EM-3)," *Proc. 10th Ann. Symp. Comp. Arch.*, (1983) 363-369

[9] Jayantha Herath; "Performance evaluation of a data-driven machine using a software simulator," *Masters Thesis, Univ. of Electrocommunications, Tokyo, Japan*, (March 1984).

[10] Arvind P., V. Kathail, K. Pingley; "A dataflow architecture with tagged tokens,"

[11] Alan L. Davis, Robert M. Keller; "Dataflow program graphs," *IEEE Computer* (1982 June), 22-41.

[12] Yoshinari Yamaguchi, Kenji Toda, Jayantha Herath, Toshitsugu Yuba, "EM-3: A Lisp Based Data-driven Machine," *FGCS*, (Nov. 1984).

[13] Jayantha Herath, K.Toda, Y.Yamaguchi, T.Yuba, "Performance Evaluation of a Multi-microprocessor Based Data-driven Machine Using a Software Simulator," *Proc. 28th Nat. Conf. IPS Japan.* (March 1984).

[14] Jayantha Herath, Saito Nobuo, Kenji Toda, Yoshinari Yamaguchi, Toshitsugi Yuba, " Not(operation): To reduce remaining packet garbage collected in dataflow computing systems," *to be published*

[15] Jayantha Herath, Hideyo Mizuba, Saito Nobuo, Kenji Toda, Yoshinari Yamaguchi, Toshitsugi Yuba, " Not(operation): An efficient unofication control mechanism to implement prolog languages in dataflow computing systems," *to be published*