PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1984,
edited by ICOT.

EM-3: A LISP-BASED DATA-DRIVEN MACHINE

Yoshinori YAMAGUCHI*, Kenji TODA*, Jayanta HERATH**, Toshitsugu YUBA*

* Electrotechnical Laboratory
1-1-4, Umesono, Sakuramura, Niiharigun
Ibaraki 305, JAPAN

** Keio University
3-14-1, Hiyoshi, Yokohama 223, JAPAN

ABSTRACT

In this paper, a Lisp-based data-driven machine with a novel parallel control mechanism and its hardware prototype are presented. The proposed control mechanism is a natural extension of the data-driven scheme to the function evaluation and is achieved by packet communication architecture.

First, the control mechanism of the ETL data-driven machine-3 (EM-3) is overviewed. Next, the architecture of its hardware prototype is described. The hardware prototype designed accommodates eight processing elements which are connected via a communication network. Each processing element consists of a microprocessor and special hardware. The network is organized by several LSI router chips. Then the instruction set of the prototype and the design philosophy of a functional programming language for the data-driven machine are presented. Finally, the performance characteristics of the machine obtained by the software simulator are evaluated.

1. INTRODUCTION

For the next generation computers, a non von Neumann computer architecture and new software environments must be developed. The new computer architecture must be based on parallel processing and VLSI technology. Data-driven architecture is proposed as a new computation model [2,4]. It has parallel processing potential in both hardware and software; i.e., the maximum inherent parallelism in ordinary programs can be exploited at the architectural level of the computers. In this case, it is not necessary to specify parallel description of a program explicitly. Moreover, functional programming and logic programming styles are suitable for data-driven architecture.

The ETL data-driven machine-3 (EM-3) is a Lisp-based data-driven machine for non-numerical computations such as symbolic manipulations involved in knowledge based information processing systems. The primary goal of the EM-3 project is to study the feasibility of the practical data-driven machine for symbol manipulations. In recent years, there have been some attempts to construct new machines with data-driven architecture. However they have shown the potential of the data-driven architecture only in principle, since those machines are too small in scale to execute large programs.

The EM-3 aims at bringing out the intrinsic parallelism in ordinary programs which are written in EMLISP [15], a functional programming language. We have proposed an advanced control mechanism for the EM-3 using novel concepts such as pseudo-result, semi-result [16] and partial-result [13]. It has been proved that these notions cause new parallelism and accelerate program execution. We have advanced the data-driven concept to meet the challenge of producing the next generation computers.

The features of the EM-3 prototype are as follows:

(1) Multi-microprocessor implementation.
(2) Attached special hardware for the packet memory.
(3) Four by four router cell fabricated by gate-array LSI.
(4) Advanced control mechanism for function evaluation.
(5) Distributed list structure.
(6) Lisp-like data-driven language.

In this paper, the new control feature adopted in the EM-3 is first described briefly. Next, the hardware organization of the EM-3 prototype, instruction set and format of the communication packets used are described. Then the Lisp-like data-driven language, EMLISP, is outlined. Finally, a performance evaluation of the EM-3 prototype using the software simulator is presented.

2. Control mechanism of the EM-3

2.1 Pseudo-results and semi-results

EM-3 is a multiprocessing system with a number of identical processing elements (PEs) in which each PE is connected via a packet communication network. Figure 1 shows the functional organization of an EM-3 processing element. The functions of the sections are described in [12]. Each section of a PE processes received packets and sends the processed packets to the designated sections.

The function evaluation mechanism of the EM-3 belongs to the class of "full substitution" [10] for recursive programs. It can be observed that eager evaluation [3,5] and incomplete objects [11] belong to a similar class of computation. The difference is that our approach is a natural extension of the data-driven scheme to function evaluation, whereas the latter two approaches are based on the reduction scheme.

In order to achieve eager evaluation on the base of a data-driven scheme, the concept of a "pseudo-result" [12,16] is introduced. In the control mechanism of the EM-3, function execution causes the generation of a pseudo-result. The pseudo-result can invoke other operations and/or defined functions as the actual-result does. When and only when all argument values for a defined function become available, a pseudo-result is generated as the virtual result of the function execution.

Advanced control and pipelining are carried out using the concept of pseudo-results which increase the concurrency in multiprocessing environments. The proposed pseudo-result scheme relaxes the firing conditions of the data-driven scheme, and therefore the evaluation of a function is advanced, and the operations contained in the function can be executed concurrently with the evaluation of its successor. This allows a certain degree of overlapping of computation.

The lenient cons mechanism [1,9] can be implemented easily using this pseudo-result scheme. The sub structure obtained by executing a cons operation may include a pseudo-result, and this sub structure is a "semi-result". A cons operation accepts any combination of actual-result, semi-result and pseudo-result as its input, and generates a semi-result. A semi-result can fire operations and/or functions. When applying car/cdr operations for a semi-result, the output, which is not necessarily an actual-result, is obtained immediately. This allows advanced computation.

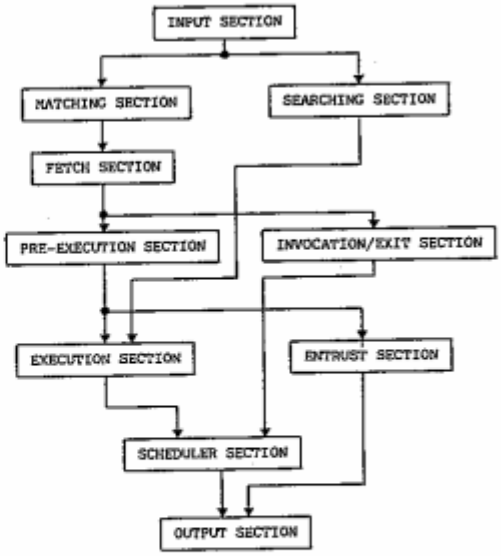

Fig. 1 Functional organization of a PE

2.2 Partial-result

The pseudo-result which is introduced in the control mechanism of the EM-3 can be regarded as a data type in the data-driven scheme. An operation manipulates a pseudo-result as an actual data. If an operation generates a new data type containing a pseudo-result, then all operations can evaluate it more eagerly. This new data type has an incomplete structure which resembles an intermediate form of the reduction mechanism. This incomplete structure is a "partial-result". In the reduction mechanism, the reduced form is rewritten by a new form until no rewriting can be applied, whereas in the partial-result mechanism, the partial-result can be manipulated freely. Therefore, if a partial-result contains any pseudo-result and if the value of this pseudo-result is a partial-result, it is possible to replace the old partial-result by a new partial-result. This process resembles the process of reduction. In the reduction mechanism, decomposition is invoked by a request, but in the partial-result mechanism, decomposition and evaluation are invoked concurrently.

Partial results are introduced here to evaluate add operation and/or multiply operation eagerly. Because the add operation and multiply operation hold the commutative law, we can calculate a series of add operation and/or multiply operations in any sequence.

The format of a partial result is assumed as follows.

(op, number-part, pseudo-result)

Here the op is + or *, which represents the operation for a partial calculation. The number-part represents the defined part of partial-result. The pseudo-result represents the undefined part of the partial-results which is designated as the value of the pseudo-result. For example, when an add operation is fired by the arrival of number 5 and pseudo-result #1, the output of the add operation will be a partial-result represented as (+, 5, #1).

We show the calculation of the partial-results by using an example. If the result of some function F is (+, 5, #1), a partial-result, and if the value of the pseudo-result part, #1 in this partial-result, is (+, 3, #2), another partial-result, then the new partial result of F is obtained by replacing #1, the pseudo-result part of the old partial-result, by (+, 3, #2). Then the present value of F is (+, 5, (+, 3, #2)) = (+, 8, #2), a new partial result. This replacement is carried out concurrently with the decomposition and generation of partial-results in the EM-3.

To execute partial calculation efficiently, two types of packets are introduced. One is a "request packet" which requests a further reduced form of a pseudo-result, and the other is a "reply packet" which gives the reply to the request packet. If the value of a reply packet is a partial-result, a new request packet for another pseudo-result, to replace the partial result, is generated. When the value of a reply packet is not a partial-result, the reduction process stops and substitutes the actual-result as the value.

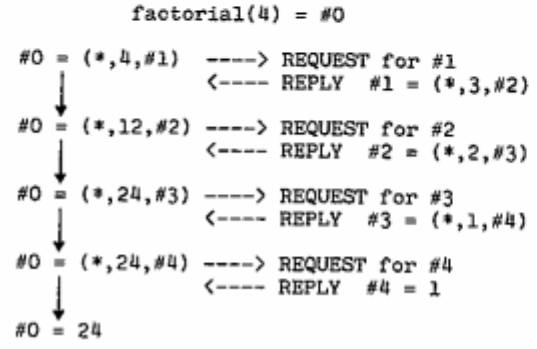

Fig. 2 Example -- partial-result calculation

Figure 2 illustrates the partial calculation of the recursively defined factorial function. It shows the computation process of the partial calculation, i.e., how the partial result is replaced and how the request or reply packet is sent or received. Note that the decomposition of the function invocation and the arithmetic operations are executed concurrently.

3. ORGANIZATION OF THE EM-3 PROTOTYPE

3.1 The structure of the EM-3 prototype

The purpose of the EM-3 prototype is
(1) To confirm the effectiveness of the EM-3 control mechanism.
(2) To measure different kinds of overheads to realize the data-driven machine.
(3) To evaluate the performance of a real data-driven machine by executing practical programs which cannot be executed on the software simulator.

Figure 3 shows the organization of the EM-3 prototype. The prototype consists of eight PEs, each of which is connected via a router network. A PE corresponds to the M68000 16-bit microprocessor with special hardware. The functions of a PE of the prototype are logically identical to those of the EM-3 which is shown in Figure 1. In the EM-3 prototype all PEs process packets in parallel, and each PE processes packets sequentially, because the function evaluation mechanism is realized by a control program in the microprocessor. The control program of the prototype is written in the C language.

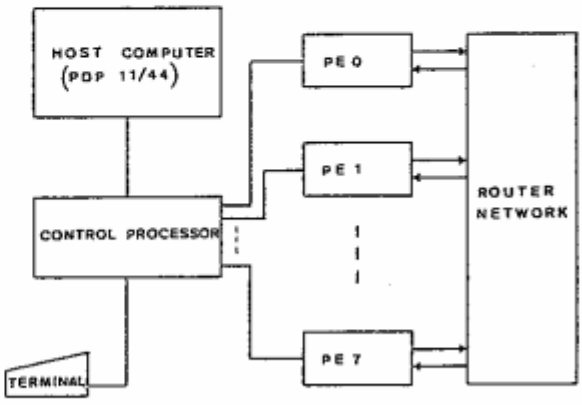

Fig. 3 Block diagram -- EM-3 prototype

There is no locality in the network. The communication packets pass from one PE to another PE via the network. Each PE is connected to the control processor via a 16-bit parallel interface. The control processor loads the user program from the host computer to all PEs, controls the status of each PE by using interruption and also controls the I/O functions.

The host computer, PDP-11/44, is used for software developments and file management. It includes the C compiler and the cross C compiler for the M68000 microprocessor. The control program of the prototype is debugged on the host computer.

Figure 4 shows the photograph of the EM-3 hardware prototype. This shows the control processor (top shelf) four PEs (each shelf contains two PEs), router network, another four PEs from top to bottom respectively. Each PE is composed of four 23cm * 25 cm boards. Router network is composed of a 35cm * 40cm board. Each PE is connected to a router network port and to the control processor by flat cables.

3.2 Organization of the PE

The organization of a PE is shown in Figure 5. The M68000 microprocessor is the main processor of a PE. The I/O interface connects the PE to the control processor. The packet memory control unit is the interface to the router network. Each unit is connected by the M68000 common bus.

Data-driven control of the EM-3 prototype is based on packet communication and packet processing. We focused our attention on the network and packet manager. The design issues of the hardware are:

(1) The communication cost of a packet must be cheaper than the cost of processing it. The packet length is rather long (maximum 224 bits) and the total packet length is divided into packet segments in the router network. Each packet is 16 bits long. Therefore the maximum number of packet segments is 14. On average the router network can transfer a packet within 2 micro seconds. This is shorter than the packet processing time. The router network is described in section 4.

(2) The processing overhead for managing the packets must be reduced. The attached hardware, packet memories and its control unit are designed for this purpose.

Fig. 4 System hardware -- EM-3 prototype

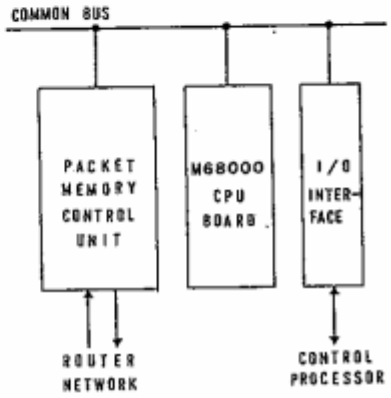

Fig. 5 Organization of a PE

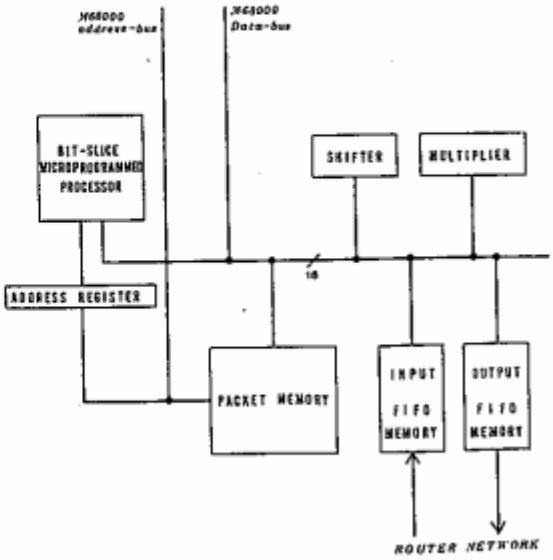

Fig. 6 Block diagram -- PMCU

Figure 6 shows the block diagram of the packet memory control unit (PMCU). This unit contains high speed memories to store communication packets. The memory size is 128 kbytes. This memory is divided into 4096 fields, so the length of each field is 256 bits. The packet memory organization is shown in the Figure 7. The first 16 bits of each field are used to implement a hash table for the matching memory. The next 16 bits are used to link the hash conflicts, link the free list of packets or link the input/output packet queue to packet communication network. The remaining 224 are bits used to store the body of the packets. Since these memories are allocated in the address space of the M68000 central processor, the processor and PMCU can read the information available in the packets or store any data to the packets. The M68000 and the PMCU run independently and concurrently. Hence it is possible for both units to access the packet memory simultaneously. In this case, preference is given to the M68000 accesses. PMCU can perform bit manipulation and simple arithmetic operations in the fields of a packet.

PMCU consist of two FIFO queues, a bit-slice microprogrammed microprocessor, an address register, a shifter and a multiplier. Input FIFO memory and output FIFO memory are used as an interface to the router network. A packet arrived from the router network is stored automatically in the input FIFO memory. A packet stored in the output FIFO memory is sent automatically to the router network. The microprogrammed processor can execute arithmetic or logical operations for the segmented fields of a packet. The shifter is used for field extraction and the multiplier is used to calculate the hash addresses. These functions are useful to reduce the overhead to process the matching function of the packets. The commands for PMCU are embedded in the address space of the main processor.

The PMCU reads the packet segments sequentially from top of the packet queue and stores then in the packet memory while associatively searching the set of packets

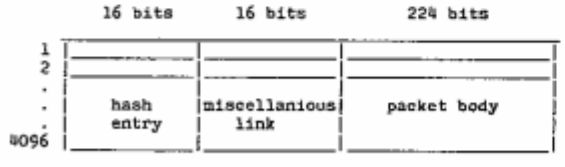

Fig. 7 Packet memory

using hashing algorithms. Free space in the packet memory is managed by a free list pointer. Packet memories are accessible from both the main processor and PMCU. Hence, the main processor can read the required data from a packet in the memories, execute them and then write the result into the packet. In the main processor, a packet is represented by a pointer to the packet memory address in the PMCU where the packet management is carried out. Hence, there is no packet movement overhead in the PE.

It is easy to add new functional units to a PE of the EM-3 prototype by simply connecting the necessary add-on hardware to the common bus. For example, a new pseudo-result control unit will be suitable for the extension of functional units of the PE.

PMCU is constructed by using about 170 MSI chips on two boards. The length of the microprogram which controls the bit-slice processor is 64 bits long and its format is almost horizontal.

3.3 Organization of the network

The router network was adopted as the communication network of the EM-3. The packet communication network is organized as a router cell network. A special LSI chip was designed for this purpose. The router cell network is a multi stage interconnection network which consists of router cell LSIs. The router cell LSI has been designed as a general purpose element in the communication network.

The router cell LSI is a store and forward matrix switch with four input and four output ports. There are two buffers in each port to transfer series of data segments successively.

The specifications of the router cell chip are as follows.

(1) The router communication chip does not have a controller in it because of the pin limitation. This gives a generalized router network. A special controller can be designed for special applications, or the same chip can be added to the general router network to achieve the control feature.

(2) Each port of the router chip sends or receives messages by four parallel bits. Any number of router cells, in four bit increments, can be interconnected to form the network.

(3) The conflict of multi port destined to a single port is treated by a priority mechanism. The priority given to each port changes according to a round robin rule.

(4) The whole router network system acts in a synchronous fashion. A central clock is supplied to all router chips and data transfers from chip to chip are executed during a single clock period.

(5) Packets are transferred through the router in pipeline fashion. A long packet is divided into packet segments and transferred as a sequence of packet segments. The end of the packet is designated by a special signal EOP.

(6) This LSI chip is composed of BI-CMOS gate array technology using 1357 gates. It can transfer one packet segment every 150 nano seconds.

The interconnection between the router cells is a modification of the delta network. The shuffle exchange network with eight input and eight output ports can comprise of two stages of four by four network cells with redundant half ports. In the implementation of the EM-3 prototype, these redundant ports are connected to implement redundant paths.

Figure 8 shows the layout for the interconnection of eight input and eight output network using two four by four router LSI chips. In the EM-3 prototype, 16 parallel bits can be dealt with using four router cells together.

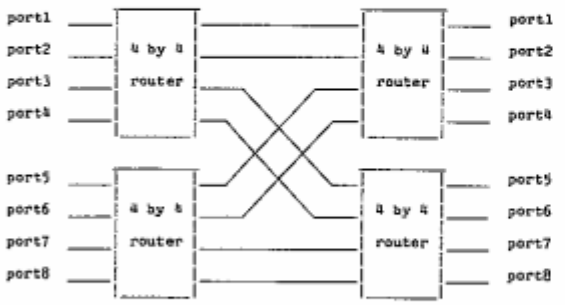

Fig. 8 Router network interconnection

Table 1 Format -- EMIL code

```
(opcode constant dest-list)
(call functionname no-of-arg no-of-ret dest-list)
(proc functionname 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
 no-of-arg ::= integer      no-of-ret ::= integer
 n ::= integer              u ::= integer
 v ::= integer              port-no ::= integer
```

4. INSTRUCTION SET

The EM-3 instruction set has a one-to-one correspondence with the intermediate language, EMIL code, used to represent data-driven graphs. Table 1 shows the format of the EMIL code. Each operation or function-name in the EMIL code represents a node of a data-driven graph and the dest-list represents the arcs of the graph. The set of data-driven operations are composed of Lisp-like primitives and a number of new operations specially constructed for the EM-3. The examples of the former operations are car, cdr, cons, atom, equal, plus, times, etc. Examples of the latter are the *distribute operation which distributes the input data to the corresponding nodes, the *switch operation which controls one input by the other input, and the *constant operation which always gives the constant data. When an operand of a operation is a constant datum, it is possible to place that constant datum in the constant field of the operation. The destination field consists of two sub fields. The label field represents the destination node name, and the other sub field represents the attribute of the node. For example, port-no represents the input port number of the operation where the operand is destinated and u in (arg u v) represent the argument number, in total of v arguments, of the function called. The instruction proc specifies the name of the function, the total number of arguments and the destination of each argument.

The EM-3 instructions are loaded to the instruction memory of all PEs by the loader. There is no instruction fetch conflict because all PEs have a copy of the instructions

5. PACKETS

A packet carries data or messages from one PE to another PE. The length of a packet varies with the packet type. There are six packet types used in the EM-3, the result packet, entrust packet, request packet, reply packet, incref packet and decref packet. The result packet carries data from one operation to the other operation and hence is used frequently in the data-driven environment. A result packet is 96 bits long; its format is shown in Figure 9. PE# represents the number of the destination PE. The PE# field is used by the router network to send a packet to the destined PE. TYPE represents the packet type number. These two fields are common to all packet types.

PSEUDO# is used to represent the environment in which the packet is generated. NODE# represents the node number of the data-driven graph for which the packet is destined. PORT represents the port number and WAIT# shows the number of inputs that can be accepted by a destination node. The OP-TAG represents miscellaneous information about the destination node, e.g. the destined node is a call instruction. The CELL field is composed of an eight bit tag field and 24 bit datum. In the router network, a result packet is divided into six 16 bit segments.

The usage of the entrust, request and reply packets are mentioned above and have different packet formats. The packet lengths are 128, 48 and 80 bits respectively. The incref packet(48 bit long) and decref packet (32 bit long) are used for collecting pseudo table and heap garbage using the reference count scheme for storage management.

6. LISP-LIKE DATA-DRIVEN LANGUAGE

All EM-3 user programs EM-3 are written in the Lisp-like data-driven language, EMLISP. EMLISP is a high level programming language based on Lisp specially designed for data-driven computations. Parallel evaluation of an EMLISP program is based on the parallel execution of the arguments in a function and is drawn naturally when an EMLISP program is translated into a data-driven graph. The syntax and basic functions of EMLISP are similar to those of Lisp. Programs represented by S-expressions must be translated into the data driven graphs easily.

The design principles of the EMLISP are as follows:
(1) Pure functionality and the single assignment rule are adopted.
(2) Global and free variables are inhibited.

Table 2 Special features -- EMLISP

[Eliminated Functions]

Relatives of prog:	PROG, PROG2, PROGN
Flow of control:	GO, DO
Modifying list:	RPLACA, RPLACD, NCONC, ...
Relatives of array:	ARRAY, STORE, ...

[Appended Functions]

Blocking:	BLOCK
Parallel cond:	PCOND

(3) The functions replacing the existing list structure are inhibited.
(4) Loop constructs are inhibited.

Table 2 shows the functions which are deleted from and added to the conventional Lisp language. The new feature "block" is introduced to EMLISP to bring a procedural programming style such as "prog" in Lisp. Bound variables must be defined first in the block, whereas S-expressions can be written in any order. The linkages between S-expressions in the block are defind by the dependencies of the variables in the S-expressions. The single assignment rule must be followed in the block structure. The "pcond" structure is introduced to bring the guarded command feature. In the pcond structure, the propositional expressions are evaluated concurrently, then the corresponding forms are evaluated when the value of the conditional expressions are true. If the conditions are mutually exclusive, there is no need to execute the single assignment rule between the forms.

7. PERFORMANCE EVALUATION BY A SOFTWARE SIMULATOR

The performance of the EM-3 prototype is evaluated using a software simulator. The software simulator simulates the program behavior faithfully. In the EM-3 prototype, the functions of a PE are logically identical to those of the EM-3, but the input packets are processed sequentially. The simulation parameters assumed for this simulation are the same as those given in [12]. A router network is used for communication.

The following benchmark programs are executed in this simulation study:
1) q(n): The parallel version of the n queen problem giving all possible solutions.
2) qs(n): The Quicksort giving O(n) parallelism with a given n data.
3) fib(n): The Fibonacci function giving binary tree parallelism.

Fibonacci contains only numerical computations. All other benchmark programs executed contain numerical and non numerical computations. The execution times required to execute above benchmark programs in the EM-3 prototype simulator are measured by varying the number of PEs and plotted in Figure 10.

The ideal data-driven parallelism is obtained by modifying the EM-3 prototype software simulator. All result packets that can be executed simultaneously are counted before the execution. This gives the number of concurrent operations. It is assumed that the execution time is the same for all operations and an unlimited number of concurrent operations can be executed in one time step. Figure 11 shows the ideal data-driven parallelism variation over the time measured for above benchmark programs. Here the horizontal axis represents the time and the vertical axis represents the ideal data-driven parallelism. The data on the vertical axis is plotted by log scale based on 2. These figures show that the EM-3 prototype extract the concurrency embedded in the programs.

8. CONCLUSION

Our goal is to construct a prototype of a Lisp-based data-driven machine to execute more practical data-driven programs and simulate a data-driven machine architecture with a large number of PEs.

In [12] we have already shown the effectiveness of the control mechanism of the EM-3 for non-numerical computations. The EM-3 prototype will confirm this at a more practical level. There are many issues to be solved on data-driven architecture. We will continue our studies on those issues in a real environment.

The first version of the PE control program was implemented. It was written in the C language and compiled to the EM-3 prototype using a cross compiler. Few bench mark programs are executed on the EM-3 prototype. All software tools except the micro program and the environmemt of the C language are coded in the C language. We are planning to evaluate and improve the architecture of the prototype by executing a number of benchmark and application programs. We believe that many ideas for data-driven architecture should be tried out on the prototype and be tested by measuring and evaluating the performance of the prototype.

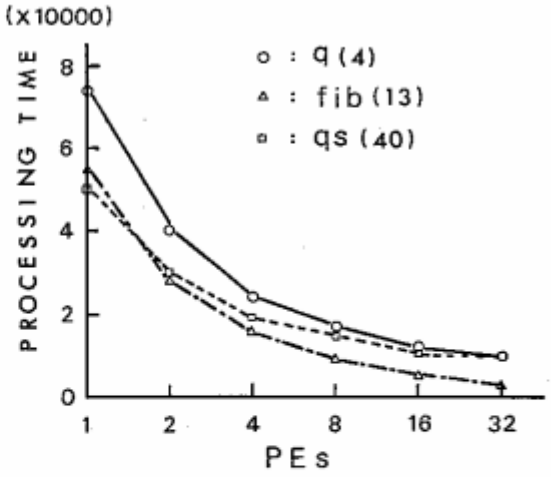

Fig. 10 Simulation results

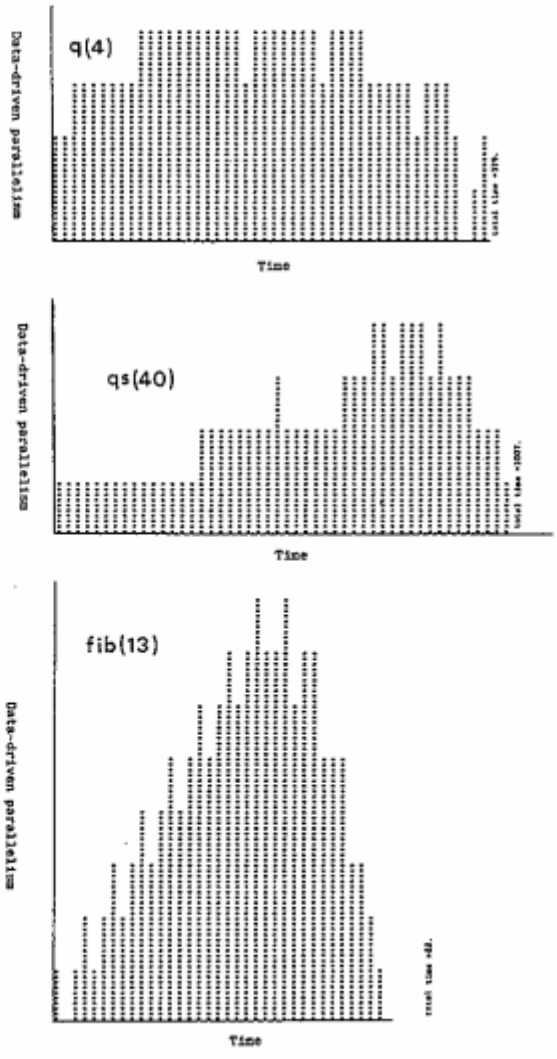

Fig. 11 Ideal data-driven parallelism variations over time

ACKNOWLEDGEMENT

We would like to thank Dr. Hiroshi Kashiwagi, Director General of Computer Systems Division, for providing the opportunity for the present study, and to the staff of Computer Architecture Section for their fruitful discussions.

REFERENCES

[1] Amamiya, M., R. Hasegawa, O. Nakamura and H. Mikami; A list-processing-oriented dataflow machine architecture, AFIPS NCC, 143-151 (1982).

[2] Arvind, V. Kathail and K. Pingali; A data flow architecture with tagged tokens, TM-174, Lab. Comp. Sci., MIT (Sept. 1980).

[3] Darlington, J. and M. Reeve; ALICE: A multi- processor reduction machine for the parallel evaluation of applicative languages, Proc. Funct. Prog. Lang. and Comp. Arch., 65-76 (Oct. 1981).

[4] Dennis, J.B., G.A. Boughton and C.K. Leung; Building blocks for data flow prototypes, Proc. 7th Ann. Symp. Comp. Arch., 1-8 (1981).

[5] Grit, D.H. and R.L. Page; Eager evaluation of functional programs and a supporting interconnection structure, Proc. 3rd Int. Dist. Comp. Sys., 811-816 (1982).

[6] Gurd, J. and I. Watson; Data driven system for high speed parallel computing -- part1: Structuring software for parallel execution, Computer Design, Vol.19, No.6, 91-100 (June 1980).

[7] Gurd, J. and I. Watson; Data driven system for high speed parallel computing -- part2: Hardware design, Computer Design, Vol.19, No.7, 97-106 (July 1980).

[8] Herat, J.; Performance evaluation of a data- driven machine using a software simulator, Masters thesis, Univ. of Electrocommunications, (March 1984).

[9] Keller, R.M., G. Lindstrom and S. Patil; A loosely-coupled applicative multi-processing system, Proc. NCC, 613-622 (1979).

[10] Manna, Z.; Mathematical Theory of Computation, McGraw-Hill, New York (1974).

[11] Peterson, J.C., W.D. Murray; Parallel computer architecture employing functional programming systems, Proc. Int. Workshop High-level Lang. Comp. Arch., 190-195 (May 1980).

[12] Yamaguchi, Y., K.Toda and T.Yuba; A performance evaluation of a Lisp-based data-driven machine (EM-3), Proc. 10th Ann. Aymp. Comp. Arch., 163 -369 (1983).

[13] Yamaguchi, Y. and T. Yuba; A partial caluculation for the numerical operations on data-driven machine EM-3, Proc. 27th Nat. Conv. IPS Japan, 7N-5 (Oct. 1983), in Japanese.

[14] Yamaguchi, Y., K. Toda and T. Yuba; The ETL data-driven machine EM-3: (1) Architecture, Proc. 26th Nat. Conv. IPS Japan, 5N-3 (Mar. 1983), in Japanese.

[15] Yuba, T., Y. Yamaguchi and T. Shimada; EMLISP: A Lisp-like language for a data-driven machine and its intermediate language, Proc. 24th Nat. Conv. IPS Japan, 7D-6 (Mar. 1982),in Japanese.

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