

A CONTROL MECHANISM OF A LISP-BASED DATA-DRIVEN MACHINE

Toshitsugu YUBA, Yoshinori YAMAGUCHI and Toshio SHIMADA

Electrotechnical Laboratory, Sakuramura, Niiharigun, Ibaraki 305, Japan

Communicated by A. Ershov

Received 15 October 1982

Keywords: Lisp, data-driven architecture, lazy evaluation, multiprocessing

1. Introduction

We are doing research on and development of a Lisp-based high level language machine with data-driven architecture, called the ETL data-driven Machine-3 (EM-3). The EM-3 is basically oriented to symbol manipulation and aims at bringing out the intrinsic parallelism in ordinary programs.

The EM-3 is a new generation Lisp machine in the sense that the basic technology is on data-driven architecture.

Recently, many types of data-driven machines have been proposed but few of these are prototyped [1]. For symbol manipulation use there are only a few plans such as that of the University of Utah [2] and the Electrical Communication Laboratory of NTT [3]. The model of the University of Colorado [4] is related to ours in some points but is independently conceived. These data-driven machines have such a 'good' property of parallelism that programming for parallel execution does not require a special language construct. A program in some data-driven language is executed in a parallel processing manner in accordance with the natural data dependencies between the operations in the program.

The design principles of the EM-3 are summarized as follows:

(1) Lisp-based machine: As a user interface a Lisp-like language called Emlisp is assumed. The Emlisp is a functional single-assignment language.

(2) Parallel evaluation: The mechanism of computation leads to parallel evaluation of 'suspension' in a lazy evaluation scheme. So the computational capability is the same as lazy evaluation, but the control of evaluation is carried out eagerly.

(3) Distribution of structure memories: Storages needed for list structures are distributed over multiple processing elements (PE). Each PE has a local store with a common addressing space.

2. Function evaluation scheme

The EM-3 has very innovative control features in order to achieve parallel lazy evaluation.

First, the notion of a 'pseudo-result' is introduced in describing parallel lazy evaluation. Function execution causes the generation of a pseudo-result in the control mechanism as shown in Fig. 1. Similar to other data-driven models, when and

F : Defined Function

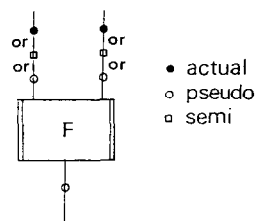


Fig. 1. Generation of a pseudo-result.

only when all argument values for the defined function F become available, the pseudo-result is generated as the virtually resulting value of the function application. At that time each argument value is not necessarily actual, as mentioned in detail later.

The notion of a pseudo-result corresponds to that of suspension in lazy evaluation. In a lazy evaluation scheme, evaluation of a computation is delayed until the following computation requires actual argument values. Whereas in pseudo-result scheme, the evaluation of a function keeps going in parallel with the evaluation of its successor. The identifiers of pseudo-results are realized by addresses on a result store and these are eventually filled by actual-results. Note that pseudo-results can fire other operations and/or defined functions as well as actual-results do.

Advanced control and pipelining are carried out by utilizing the notion of pseudo-results and they may increase concurrency in multiprocessing environments. The introduction of pseudo-results relaxes firing conditions 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 some degree of overlapping of computation.

Secondly, the operation cons in Emlisp constructs either a 'semi-result' or an actual-result as illustrated in Fig. 2. A cons accepts any combination of actual-results, semi-results and pseudo-results as its inputs, and then generates a semi-result. Like a list cell of a conventional Lisp, a semi-result has car and cdr parts as its substructure. Semi-results are able to fire operations/functions as well. In case of applying car/cdr operations for a semi-

λ : CONS

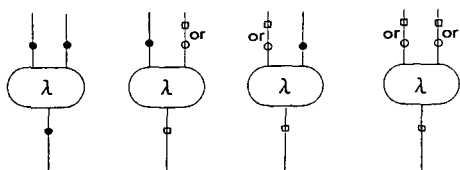


Fig. 2. Generation of a semi-result in a cons operation.

μ : CAR or CDR

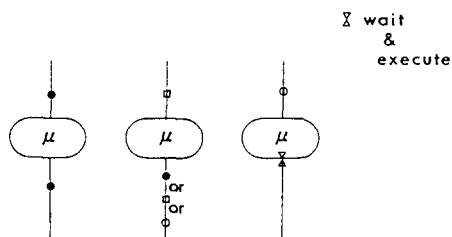


Fig. 3. Application of a car/cdr operation.

result, the output is immediately obtained but it is not always an actual-result.

In list operations, the type of their outputs differs from each other depending on the individual case, as shown in Figs. 3 and 4. For example, car and cdr operations may be immediately executed when the input is an actual-result or a semi-result. When a structure result arrives at an atom or a null operation, the output becomes an actual-result ('false'). If there is no pseudo-result at inputs, the operation eq can immediately generate an actual-result. But when the inputs to list operations include a pseudo-result, the execution is deferred until its actual value arrives.

Third, for non-list operations like arithmetic, the execution of operations will be deferred until the inputs turn into actual-results. Fig. 5 shows the general operation with one and two inputs. When an operation is deferred, the control mechanism works as follows: An 'entrust' packet, whose format is given later, is generated and sent to another PE according to the pseudo-result identifier. There, the operation waits until the pseudo-result has been turned into an actual-result.

θ : ATOM or NULL

ω : EQ

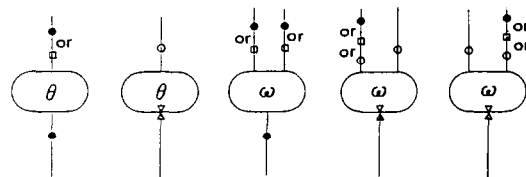


Fig. 4. Application of an atom, null or eq operation.

P, Q : General Operation

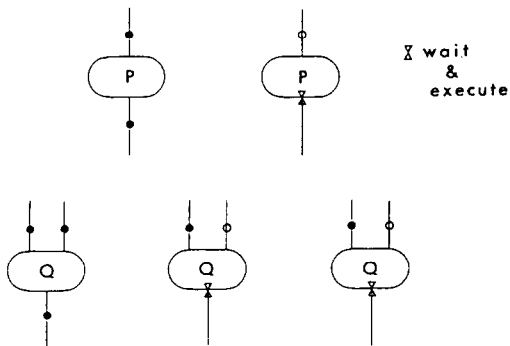


Fig. 5. Application of a general operation.

3. Functional organization of a processing element

The functional diagram of the PE of the EM-3 is illustrated in Fig. 6. Each PE is assumed to be connected together through communication lines such as buses, routers or crossbar switches.

The Input Section receives three types of packets; these are 'result', 'invoke' and 'entrust' packets which are shown in the next section in concrete form. These are placed into a queue. Each packet in the queue is sent to a corresponding section according to its type. Outputs of each section are sent to another section of the same PE in a pipeline manner, or are placed in a queue of the Output Section. Packets on the output queue are sent to another PE.

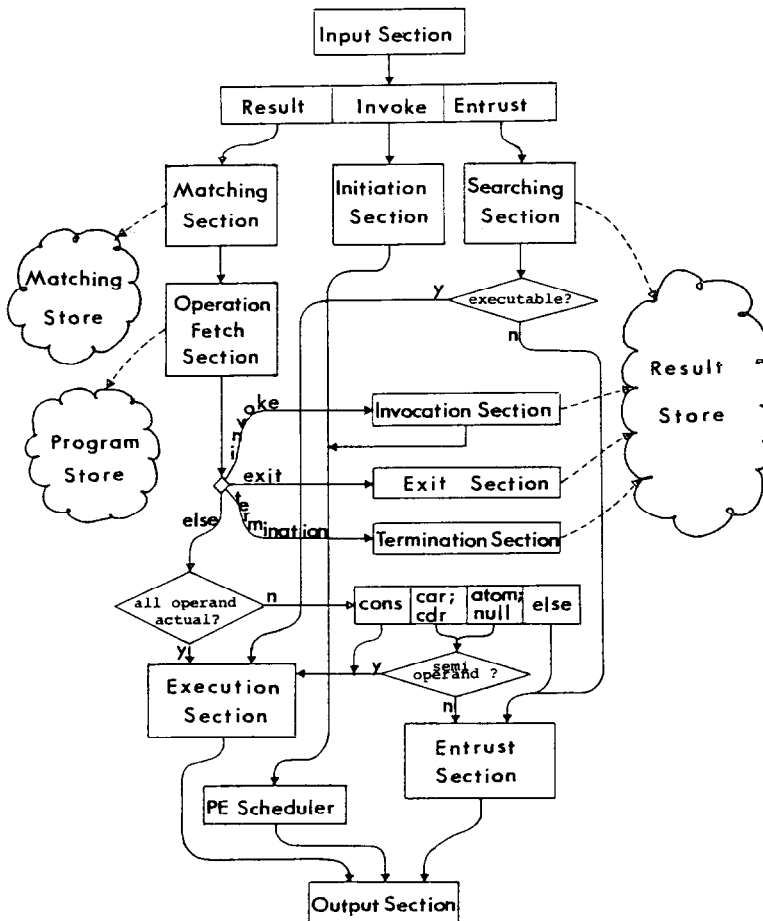


Fig. 6. Functional organization of the PE.

In the Operand Matching Section, packets destined to operands of an operation and/or a defined function are matched with each other in the Matching Store and then synchronized as a consequence. Unmatched packets are in a waiting state and are stored in the Matching Store, which is associatively searched whenever needed.

The Operation Fetch Section fetches operations from the Program Store according to the operation addresses sent from the Matching Store and then combines them with their operands to generate internal packets.

The Initiation Section accepts invoke packets and extracts a function name and its arguments placed in the packets, and then generates result packets corresponding to each argument. After that, the body of the function is activated by these result packets.

The Invocation Section works when the operation fetched at the Operation Fetch Section is an invoke operation which invokes a defined function. A pseudo-result identifier is created for a newly invoked function and, at the same time, an invoke packet is generated and sent to the PE scheduler.

The Searching Section is activated by an entrust packet. An entrust packet is associated with a pseudo-result identifier whose actual value is required. Using this identifier, the Pseudo-result Table in the Result Store is searched. If the corresponding actual result is found and the entrust packet is immediately executable, i.e., its operands are sufficiently complete to allow execution of the operation, then it is sent to the Execution Section. If not found, the entrust packet is stored at the Deferred Buffer in the Result Store and waits for completion of the predecessor operation assigned by the pseudo-result identifier.

At the exit of each function, the Exit Section saves values of actual-results or pointers to semi-results which correspond to each pseudo-result of the function into the Pseudo-result Table. This time, the entrust packets waiting for completion of the function corresponding to the exit operation are activated and, if immediately executable, they are then sent to the Execution Section or else to the Entrust Section.

The Entrust Section generates an entrust packet

when inputs include a pseudo-result, and defers the execution of the operation. The generated entrust packet is sent to a PE which is assigned by the pseudo-result identifier.

Note that the assignment of a pseudo-result identifier is carried out at each PE independently. Therefore, the global identifier can be represented by combining a pseudo-result identifier with the number of the PE where it is created.

4. Formats of packets

The following are the formats of each packet:

Result Packet:

⟨'result', PE, color, destination, value⟩.

Invoke Packet:

⟨'invoke', PE, color, function-name, arguments, dest-list⟩.

Entrust Packet:

⟨'entrust', PE, color, opcode, operands, dest-list⟩.

In a result packet, PE represents the number of a PE to which the result packet is sent and the color means a tag discriminating the body of a function which is invoked recursively. The destination points to the successor operation and is represented by an operation address, a 1-bit number indicating the port of the successor operation and the number of inputs to enable the destination address.

In an invoke packet, the function-name is the name of an invoked function and the arguments field is its parameters. The dest-list is a list of destinations mentioned above. The opcode and operands in an entrust packet are the code of an entrusted operation and its operand values, respectively.

5. Conclusion

In this paper a model of the control mechanism for a Lisp-based data-driven machine is proposed and the functional configuration of a processing element is shown on the base of the model. By the

introduction of the notions pseudo-result and semi-result, it is shown that 'parallel' lazy evaluation can be naturally realized in data-driven architecture.

In conclusion we present an overview of current related research; one of the studies currently in progress is the measurement of intrinsic parallelism in Lisp programs. A summary of this empirical study was reported in [6]. The experimental results will be useful for designing a Lisp-like language which should be well suited to data-driven architecture and for estimating the sizes of each store in a PE and the communication traffic among PEs.

Another study is the design of a user interface language for the EM-3, Emlisp, and the implementation of its compiler. We have assumed an intermediate language for data-driven architecture, named Emil, which is useful for describing a data-flow graph. The Emlisp compiler translates an Emlisp program into an Emil program.

A further study is the development of a software simulator for EM-3's processing elements. This work is now being carried out in order to confirm the EM-3 control mechanism and to evaluate its efficiency. This simulator accepts an Emil program as its source. Written in SIMULA, it is able to simulate the pipeline behavior of the

EM-3 through defining each PE section as a 'class'.

We believe that many ideas about data-driven architecture should be prototyped and then the feasibility of implementation should be tested by measuring and evaluating performance of the prototypes.

References

- [1] V.K. Avind and K. Pingali, A data flow architecture with tagged tokens, Rept. TM-174, Lab. for Computer Science, MIT, 1980.
- [2] R.M. Keller, G. Lindstrom and S. Patil, A loosely-coupled applicative multi-processing system, Proc. NCC (1979) 613-622.
- [3] M. Amamiya, R. Hasegawa and H. Mikami, A list processing oriented data flow machine and its simulator, Papers of Tech. Group on Comp. Arch. 40-8, IPS Japan (1981) 77-88 (in Japanese).
- [4] J.C. Peterson and W.D. Murrey, Parallel computer architecture employing functional programming systems, Proc. Internat. Workshop Highlevel Language Computer Architecture (1980) 190-195.
- [5] Y. Yamaguchi, T. Yuba and T. Shimada, A function evaluation mechanism of EM-3, Papers of Tech. Group on Comput. EC81-57, IECE Japan (1981) 1-8 (in Japanese).
- [6] Y. Yamaguchi, M. Yamada and T. Yuba, An empirical study of intrinsic parallelism in Lisp programs, Proc. 24th Nat. Conf. of IPS Japan (1982) 101-102 (in Japanese).