Data Flow Computers

Article · November 2012

CITATIONS
0

READS
1,582

4 authors, including:

Pramila M. Chawan
Veermata Jijabai Technological Institute
110 PUBLICATIONS **195** CITATIONS

SEE PROFILE

International Journal of Networking & Parallel Computing
Volume 1, Issue 2, November, 2012

Data Flow Computers

Ajinkya Gadkari
Student at Department of Computer Technology, Veermata Jijabai Technological Institute

Shrijit Pandey
Student at Department of Computer Technology, Veermata Jijabai Technological Institute

Hari Har
Student at Department of Computer Technology, Veermata Jijabai Technological Institute

P.M. Chawan
Professor at Department of Computer Technology, Veermata Jijabai Technological Institute

Abstract

This paper describes about Data Flow Computers. The dataflow model of computation offers an attractive alternative to control flow in extracting parallelism from programs. The execution of a dataflow instruction is based on the availability of its operand(s); hence, the synchronization of parallel activities is implicit in the dataflow model. Instructions in the dataflow model do not impose any constraints on sequencing except for the data dependencies in the program.

Keywords

Dataflow Computers, Dataflow graphs.

1. INTRODUCTION

Data flow computers are based on the concept of data-driven computation, which is drastically different from the operation of conventional von Neumann machine. The fundamental difference is that instruction execution in a conventional computer is under program-flow control, whereas that in a data flow computer is driven by the data (operand) availability.

However, studies have revealed a number of inefficiencies in dataflow computing: the data flow model incurs more overhead during an instruction cycle compared with its control-flow counterpart, the detection of enabled instructions and the construction of result tokens generally will result in poor performance for applications with low degrees of parallelism, and the execution of an instruction involves consuming tokens on the input arcs and generating result token(s) at the output arc(s), which involves communication of tokens among instructions. In this paper addresses the various issues and the developments in dataflow computing.

The data-driven concept means asynchrony, which means that many instructions can be executed simultaneously and asynchronously. A higher degree of implicit parallelism is expected in dataflow computer. Becuase there is no use of shared memory cells, dataflow programs are free from side effects.

This paper is organized as follows: the Dataflow Principles section reviews the basic principles of the dataflow model. The Dataflow Graphs section gives the representations used in dataflow system. The Dataflow Architectures section provides a general description of the dataflow architecture. The discussion includes a comparison of the architectural characteristics and the evolutionary improvements in dataflow computing.

2. DATAFLOW PRINCIPLES

The data flow model deals only with values and not with names of value containers (i.e., addresses). This concept is also fundamental to purely functional or applicative languages, and hence these languages have no built-in notion of global updatable memory. An operator in these languages produces a value which is used by other operators. The data flow model has nothing like an instruction counter: an instruction is enabled if and only if all the required input values have been computed. Enabled instructions consume input values, execute, and produce sets of output values which are sent to other instructions that need these values. An instruction in data flow does not introduce sequencing constraints other than the ones imposed by data dependencies in the algorithm. Thus it is possible to expose all of the parallelism in a data flow program.

A program in a high-level data flow language is directly translatable into a graph whose nodes represent functions and whose arcs represent data dependencies between functions. In the graph values are represented as tokens on the arcs. The advantage of data flow languages is that the graph can be generated with little analysis.

Data flow processors are stored-program computers in which the stored program is a representation of data flow graphs. The processor itself is designed to recognize which of the instructions in its program memory are enabled, and all such instructions are dispatched to execution units as soon as resources are available. There is no notion of a single locus of control. Any two instructions in the program memory may be executed concurrently. If sufficient resources are provided, the processor can exploit all concurrency present in the program.

Features in the data flow model:

- Intermediate or final results are passed directly as data token between instructions.
- There is no concept of shared data storage as embodied the traditional notation of a variable.
- Program sequencing is constrained only by data dependency among instructions.

3. DATAFLOW GRAPHS

Dataflow graphs can be viewed as the machine language for dataflow computers. A data flow graph is a directed graph whose nodes correspond to operators and arcs are pointers for forwardig data tokens. The graph demonstrates sequencing constraints among instructions. In data flow computer, the machine level program is represented by data flow graphs. In a dataflow computer, a program isn't represented by a linear instruction sequence, but by a dataflow graph. Moreover, no single thread of control moves from instruction to instruction demanding data, operating on it, and producing new data. Rather, data flows to instructions, causing evaluation to occur as soon as all operands are available. Instructions are known as nodes, and instead of data items one talks of tokens. A producing node is connected to a consuming node by an arc, and the "point" where an arc enters a node is called an input port. The execution of an instruction is called the firing of a node. This can only occur if the node is enabled, which is determined by the enabling rule. Usually a strict enabling rule is specified, which states that a node is enabled only when each input port contains a token. Data is sent along the arcs of the dataflow graph in the form of tokens, which are created by computational nodes and placed on output arcs. They are removed from the arcs when they are accessed as input by other computational nodes. Concurrent execution is a natural result of the fact that many tokens can be on the dataflow graph at any time; the only constraint on evaluation order is the presence of tokens on arcs in the graph. The below figure illustrates an example of dataflow graph for evaluation of expression $X^2 - 2*X + 3$. Nodes represent the operation to be performed. The fanout of an arc from a node, such as the phantom node X, denotes the conceptual replication of tokens leaving that node. A node marked with a constant value is assumed to regenerate that value as often as it is needed by nodes to which it is input. When value of X is available its provided to calculate X^2 and 2*X. The following subtraction operation will not be carried out until these values are available. As soon as X^2 and 2*X values are computed subtraction operation will be carried out which in turn will provide input to next addition operation.

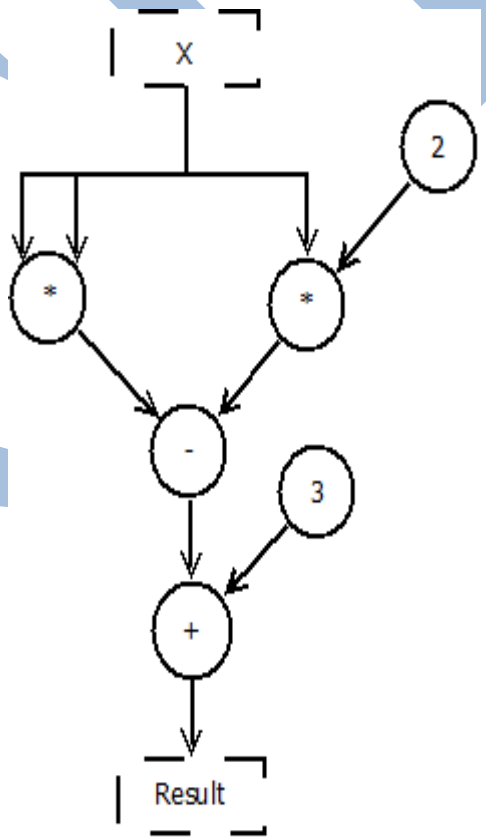

Figure 1. Data flow graph for $X^2 - 2*X + 3$.

4. DATAFLOW ARCHITECTURE

Depending on the way of handling data tokens, data flow computers are divided into static model and dynamic model. The static approach allows at most one instance of a node to be enabled for firing, i.e., a dataflow actor can be executed only when all of its input tokens are available and no tokens exist on any of its output arcs.On the other hand, the dynamic approach permits simultaneous activation of several instances of a node during the run-time by viewing arcs as buffers containing multiple data items. To distinguish between different instances of a node (and routing data for different instantiations of the node), a tag is associated with each token that identifies the context in which a particular token was generated. An actor is considered executable when all of its input tokens with identical tags are available. Static and dynamic dataflow computer organizations are as shown in below figures.

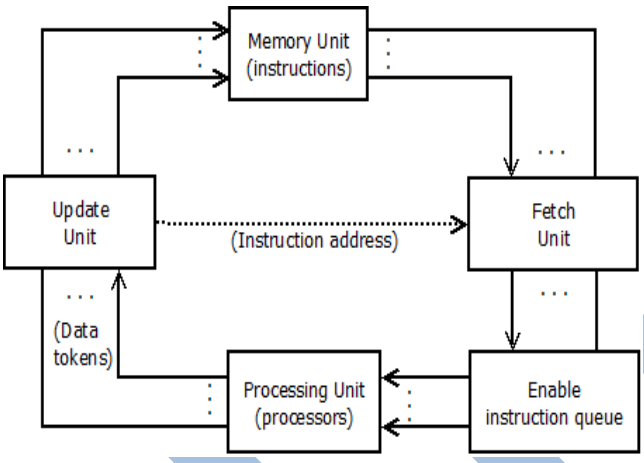

Figure 2. Static Dataflow Computer Organisation

In static data flow machine, data tokens are assumed to move along the arcs of the data flow program graph to the operator nodes. The nodal operation gets executed when all its operand data are present at the input arcs. Only one token is allowed to exist on any are at any given time, otherwise the successive sets of tokens cannot be distinguished. This architecture is considered static because tokens are not labeled and control tokens must be used to acknowledge the proper timing in transferring data tokens from node to node.

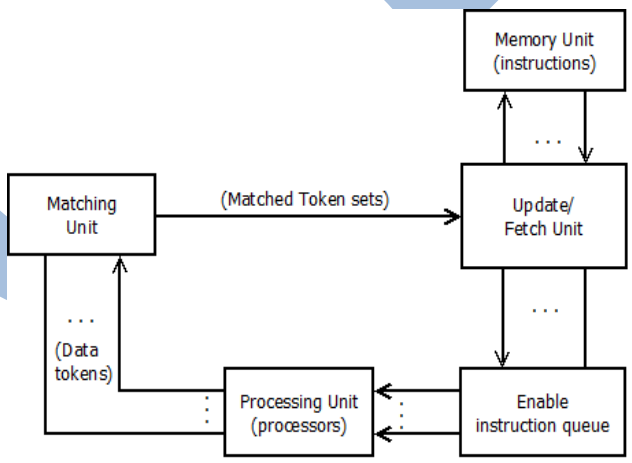

Figure 3. Dynamic Dataflow Computer Organisation

A dynamic data flow machine uses tagged tokens, so that more than one token can exist in an arc. The tagging is achieved by attaching a label with each token which uniquely identifies the context of that particular token. This dynamically tagged data flow model suggests that maximum parallelism can be exploited from a program graph. If the graph is cyclic, the tagging allows dynamically unfolding of the iterative computations. The static dataflow model has a simplified inherent mechanism to detect enabled nodes, but the model limits the performance because iterations are executed one at a time. The dynamic dataflow allows

greater exploitation of parallelism; however, this advantage comes at the expense of the overhead in terms of the generation of tags, larger data tokens, and complexity of the matching tokens.

5. EARLIER DATAFLOW ARCHITECTURES

This section discusses two classic dataflow machines: the static dataflow machine and the (dynamic) Manchester machine. These projects represent the pioneering work in the area of dataflow. The foundation they provide has inspired many other dataflow projects.

4.1. Dennis Static Dataflow Machine:

The memory section is a collection of memory cells, each cell composed of three memory words that represent an instruction template. The first word of each instruction cell contains op-code and destination address(es), and the next two words represent the operands. The design has envisioned six types of templates that represent binary and unary operators, binary and unary deciders (predicates), and binary and unary Boolean operators. The processing section is composed of five pipelined functional units, which perform the operations, form the result packet(s), and send the result token(s) to the memory section. The arbitration network is intended to establish a smooth flow of enabled instructions (i.e., instruction packet) from the memory section to the processing section. An instruction packet contains the corresponding op-code, operand value(s), and destination address(es). The distribution network is intended to transfer the result packets from the processing section to the memory section. Finally, the control network in designed to reduce the load on the distribution network by transferring the Boolean tokens and the acknowledgement signals from the processing section to the memory section.

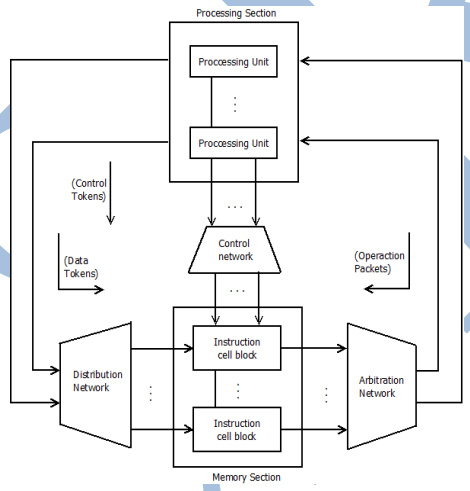

Figure 4. Dennis Static Dataflow Machine

Instructions held in the memory section are enabled for execution by the arrival of their operands in data tokens from the distribution network and control tokens from the control network. Enabled instructions, together with their operands, are sent as operation packets to the processing section through the arbitration network. The results of instruction execution are sent through the distribution network and the control network to the memory section where they become operands of other instructions. Each instruction cell has a unique address, the cell identifier. An occupied cell holds an instruction consisting of an operation code and several destinations. Each destination contains a destination address, which is a cell identifier, and additional control information used by processing units to generate result tokens. An instruction represents one or more operators of the program graph, together with its output links. Instructions are linked together through destination addresses stored in their destination fields.

Each instruction cell contains receivers which await the arrival of token values for use as operands by the instruction. Once an instruction cell has received the necessary operand tokens and acknowledge signals, the cell becomes enabled and sends an operation packet consisting the instruction and the operand values to the appropriate processing unit through the arbitration network. Note that the acknowledge signals are used to correctly implement the firing rule for program graphs.

The arbitration network provides a path from each instruction cell to each processing unit and sorts the operation packets among its output ports according to the operation codes of the instructions they contain. For each operation packet received, a processing unit performs the operation specified by the instruction using the operand values in the packet and produces one or more result tokens, which are sent to instruction cells through the control network and distribution network, Each result token

consists of at result value and a destination address derived from the instruction being processed by the processing unit. There are control tokens containing boolean values or acknowledge signals, which are sent through the control network, and data packets containing integer or complex values, which are sent through the distribution network. The two networks deliver result tokens to receivers of instruction cells as specified by their destination address fields; that is, data packets are routed according to their destination address. The arrival of a result token at an instruction cell either provides one of the receivers of the cell with an operand value or delivers an acknowledge signal; if all result tokens required by the instruction in the cell have been received, the instruction cell becomes enabled and dispatches its contents to the arbitration network as a new operation packet.

The functions performed by the processing unit are distributed among several sections of the data flow processor. The operations specified by instructions are carried out in the processing section, but control of instruction sequencing is 21 function of the control network, and the decoding of operation codes is partially done within the arbitration network. The address fields (destination addresses) of instructions specify where the results should be sent instead of addressing a shared memory cell. Instead of instructions fetching their operands, the operand values are sent to the instructions.

All communication between subsystems in the Dennis machine is by packet transmission over the channels. The transmission of packets over each channel uses an asynchronous protocol so that the five sections of the computer can operate independently without using central timing signals. Systems organized to operate in this manner are said to have the packet communication architecture.

The instruction cells are assumed to be physically independent so at any time many of them may be enabled. The arbitration network should be designed to allow many instruction packets to flow through it concurrently. Similarly, the control network and the distribution network should be designed to distribute dense streams of control and data packets back to the instruction cells. In this way, both the appetites of pipelining and parallelism are satisfied. The arbitration, distribution, and control networks of the data flow processor are examples of packet-switched routing networks that perform the function of directing packets to many functional units of the processor. If the parallelism represented in the data flow graph is to be fully exploited, routing networks must have a high bandwidth.

4.2. Manchester Dynamic dataflow Machine:

Manchester Machine is designed as a backend, composed of five units organized as a pipeline ring: The switch unit establishes communication between the frontend and back-end processor, and routes the result tokens back to the pipeline ring. The token queue is a First-in-first-out buffer that stores temporarily tokens traversing on the data-flow graph arcs. The basic operation of the matching unit is to bring together tokens with identical tags by pairing associatively tokens with the same destination node address and context. The dataflow program that represents the code for an operation is stored in the node store. The processing unit, a micro-programmed, 2-stage pipeline unit, executes the dataflow operations. The first stage handles the generation of result tokens and the association of tags with tokens. The second pipeline stage consists of 15 functional units to perform the necessary operations.

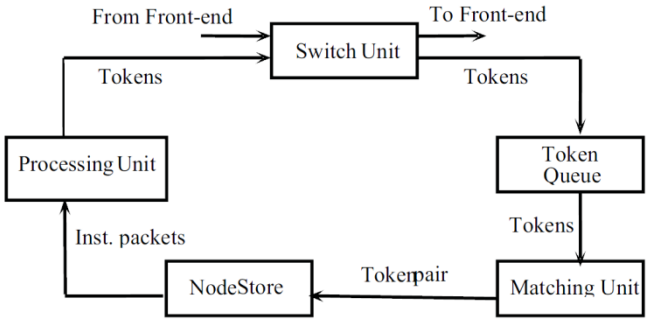

Figure 5. The Manchester Dynamic dataflow Machine

6. Conclusion

The advances from the development of dataflow machines indicate potential high performance computation based on the dataflow principles. This is necessary owing to increased demands of processing of complex scientific and technical data. As such applications require large processing times, data flow computers may help reduce the processing times and thus improve efficiency and effectiveness of implemented algorithms. However, there are still many issues to be addressed for the efficient use of dataflow computers.

7. REFERENCES

[1] Kai Hwang, Faye A. Briggs, “Computer Architecture and Parallel Processing” McGraw Hill, 1985.
[2] "DATAFLOW COMPUTERS: THEIR HISTORY AND FUTURE", Wiley Encyclopaedia of Computer Science and Engineering, edited by Benjamin Wah.
[3] ARTHUR H. VEEN, "Dataflow Machine Architecture", ACM Computing Surveys, Vol. 18, No. 4, December 1986.
[4] Alan L. Davis and Robert M. Keller, "Data Flow Program Graphs", IEEE Computer 15.2 (February 1982): 26-41. DOI: 10.1109/MC.1982.1653939
[5] Walid A. Najjara, Edward A. Leeb, Guang R. Gaoc, "Advances in the data¯ow computational model", Parallel Computing 25 (1999) 1907-1929.
[6] J. R. Gurd, C. C. Kirkham, and I. Watson, “The manchester prototype data-flow computer”, Comm. ACM, 28 (1): 34–52, 1985.
[7] J.B. Dennis, “The Evolution of ’Static’ Data-Flow Architecture” Advanced Topics in Data-Flow Computing, J.-L. Gaudiot and L. Bic, eds., Prentice Hall, 1991.