METHOD FOR CONCURRENT EXECUTION OF PRIMITIVE OPERATIONS BY DYNAMICALLY ASSIGNING OPERATIONS BASED UPON COMPUTATIONAL MARKED GRAPH AND AVAILABILITY OF DATA

Inventors: John W. Stoughton; Roland V. Mielke, both of Virginia Beach, Va.

Assignee: Center for Innovative Technology, Herndon, Va.

Filed: Mar. 24, 1987

ABSTRACT

Computationally complex primitive operations of an algorithm are executed concurrently in a plurality of functional units under the control of an assignment manager. The algorithm is preferably defined as a computationally marked graph containing data status edges (paths) corresponding to each of the data flow edges. The assignment manager assigns primitive operations to the functional units and monitors completion of the primitive operations to determine data availability using the computational marked graph of the algorithm. All data accessing of the primitive operations is performed by the functional units independently of the assignment manager.

2 Claims, 9 Drawing Sheets
FIG. 1

INPUT NODE

DATA

MULTIPLICATION NODE

DATA

ADDITION NODE

DATA

OUTPUT NODE

CONSTANT A

NODE SUPPLYING CONSTANT A

DATA

MULTIPLICATION NODE

CONSTANT B

DATA

MULTIPLICATION NODE

DATA

\[ X_n = b \cdot x_{n-1} + a \cdot u_n \]
FIG. 5

ALGORITHM LOADER 102

ASSIGNMENT MANAGER 104

TOKEN BUS 110

FUNCTIONAL UNIT # 1 105
FUNCTIONAL UNIT # 2 106
FUNCTIONAL UNIT # K 107

DATA BUS 112

GLOBAL DATA MEMORY 108

EXTERNAL DATA 114
FIG. 6

START

INPUT ALGORITHM DEFINED AS PRIMITIVE OPERATIONS AND DATA FLOW IN BETWEEN

PROVIDE DEFINITIONS OF ALGORITHMS AND PRIMITIVE OPERATIONS TO ASSIGNMENT MANAGER

ASSIGN PRIMITIVE OPERATION TO AVAILABLE FUNCTION UNIT

DATA AVAILABLE?

NO

ALGORITHM COMPLETED?

NO

YES

STOP
METHOD FOR CONCURRENT EXECUTION OF
PRIMITIVE OPERATIONS BY DYNAMICALLY
ASSIGNING OPERATIONS BASED UPON
COMPUTATIONAL MARKED GRAPH AND
AVAILABILITY OF DATA

ORIGIN OF THE INVENTION
The invention described herein was developed in part
with U.S. Government support under contract NASA-
17993 with the National Aeronautic and Space Admin-
istration (NASA). The U.S. Government has certain
rights in this invention.

BACKGROUND OF THE INVENTION
1. Field of the Invention
The present invention is related to the analysis of the
concurrently processed computationally complex algo-
rithm and, more particularly, to a data processing sys-
tem that uses a Petri net model of concurrently pro-
cessed computationally complex algorithms.

2. Description of the Related Art
One method which is currently being developed to
to increase the execution speed of computers is parallel
processing of primitive operations in an algorithm. The
hardware used in a data processing system having such
a parallel architecture is relatively easy to develop by
using, e.g., identical, special purpose computing ele-
ments each of which can access a shared memory. In
comparison, it has been much more difficult to develop
the software required to schedule, coordinate and com-
 municate between the individual computing elements
and other portions of the data processing system, such
as external data input/output to and from terminals,
printers, tape drives, etc.

Conventional multiprocessors typically execute
a program in a single processor, while different
program(s) executes in the other processor(s). The
scheduling, coordination and communication problems
in such multiprocessors are among the most
complex addressed by existing computer systems. How-
ever, in the type of parallel processing discussed above,
solution of coordination and communication problems
is even more critical, because a single program or algo-

rithm is being processed by more than one computing
element. As a result, in addition to sharing the hard-
ware, as in the conventional multiprocessor system, the
computing elements also share data. Since a first
instruction or task may be performed in a first comput-
ing element to generate intermediate data used in a second
task performed by a second computing element, the
flow of data in the system must be carefully controlled.

Such data processing systems which referred to as per-
forming concurrent parallel processing and may have a
data-driven architecture, i.e., controlled by data flow,
or a demand-driven architecture.

The differences between a data processing system
performing concurrent operations and a system orga-
nized according to von Neumann principles render
conventional methods of describing and defining com-
puter operation inadequate. For example, flowcharts or
conventional algorithm graphs which are useful in de-
scribing the operation of a von Neumann structure are
insufficiently descriptive of the operation of a concur-
rent processing system. Techniques for defining and
controlling the operation of a concurrent processing
system is critical for the development of software to be
executed on data processing systems having, e.g., a
data-driven architecture.

Efforts have been made to define tasks to be per-
formed in a data processing system for concurrent exe-
cution of primitive operations of an algorithm. For
example, previous proposals for assignment of the com-
puting elements or functional units to particular tasks
have used either static or dynamic assignment. When
static assignment is used, the tasks performed by each
functional unit is determined during the design of a
program. This requires a great deal of specificity in the
program, increasing the amount of time required for
development and thereby reducing the benefits gained
from using data flow control in a processing system.

Dynamic assignment, on the other hand, determines the
operation performed by each functional unit at the beginning
of execution of an algorithm. This reduces the effort
required during programming, but still results in fixed
corependence between a functional unit and a primiti-
ve operation during the processing of the algorithm.

SUMMARY OF THE INVENTION
An object of the present invention is to provide a
data-driven architecture in which functional units are
continuously assigned during concurrent execution of
primitive operations of an algorithm.

Another object of the present invention is to provide
scheduling, coordination and communication for a plu-
arity of functional units concurrently executing compu-
tationally intensive primitive operations of an algorithm
by providing a controlled path for each data path in a
Petri net model of the algorithm.

Another object of the present invention is to provide
a data processing system having multiple functional
units concurrently processing computationally inten-
sive primitive operations of an algorithm in which the
operations performed by the functional units are contin-
uously reassigned during execution of the algorithm.

Yet another object of the present invention is to pro-
vide a concurrent data processing system in which con-
currency is maximized.

A further object of the present invention is to provide
a data processing system having a plurality of functional
units concurrently executing primitive operations of an
algorithm which is slowed, but not prevented from
executing an algorithm, if one or more, but not all,
functional units become inoperative.

The above objects are achieved by providing a
method for concurrent execution of primitive opera-
tions of an algorithm in a data processing system having
a data-driven architecture comprising an assignment
manager, a plurality of functional units connected to the
assignment manager and a global data memory con-
ected to the functional units. The method comprises
the steps of defining an algorithm in terms of primitive
operations and data flow between the primitive opera-
tions; providing definitions of the algorithm and the
primitive operations to the assignment manager; assign-
ing each primitive operation to an available function
unit in dependence upon the definitions previously de-

defined as the data for that primitive operation becomes
available; and monitoring completion of each primitive
operation to determine data availability.

These objects, together with other objects and advan-
tages which will subsequently apparent, reside in the
details of construction and operation as more fully here-
inafter described and claimed, reference being had to the
accompanying drawings forming a part hereof,
wherein the like reference numerals refer to like parts throughout.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an example of an algorithm directed graph; FIG. 2 is a node marked graph for a node in an algorithm marked graph having two inputs and two outputs; FIG. 3A is the node marked graph of FIG. 2 with tokens representing the idle state; FIG. 3B is the node marked graph of FIG. 2 with tokens representing the process ready state; FIG. 3C is the node marked graph of FIG. 2 with tokens representing the data accepted state; FIG. 3D is the node marked graph of FIG. 2 with tokens representing the processing function - outputs empty state; FIG. 3E is the node marked graph of FIG. 2 with tokens representing the process complete - outputs delivered state; FIG. 3F is the node marked graph of FIG. 2 with tokens representing the process complete - inputs available state; FIG. 3G is the node marked graph of FIG. 2 with tokens representing the data consumed state; FIG. 3H is the node marked graph of FIG. 2 with tokens representing the process complete - outputs delivered full state;

FIG. 4 is a computational marked graph corresponding to the algorithm marked graph in FIG. 1.

FIG. 5 is a block diagram of a parallel processing system to which the present invention can be applied.

FIG. 6 is a flowchart of a method according to the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

As discussed above, at the present time it is much easier to develop hardware for parallel processing than to develop software which takes advantage of such hardware to execute a single algorithm. While it is possible to specify the operations to be performed in each processing unit in a parallel processor, this requires a great deal of programming time and also requires that the specified number of processors be available at run time while making no use of any additional processors which might be available. This is referred to as static assignment of processing or functional units. Dynamic assignment of functional units at the time processing begins is only a slight improvement over static assignment in that it is difficult to compensate for a failure of a processing unit during execution of an algorithm. As a result, there is little fault tolerance in such systems and a great deal of programming effort is required.

According to the present invention, these obstacles are overcome by decomposing an algorithm into a sequence of primitive operations, some of which can be performed concurrently. Preferably, the algorithm is represented as a Petri net to clearly display both data and control flow in the execution of the algorithm. The decomposition process is performed in the three steps which follow. Each step will be described in more detail below.

First, according to a preferred embodiment of the present invention, an algorithm graph is developed which depicts primitive operations of the algorithm and the data flow between the primitive operations. Equation (1) is an example of a relatively simple algorithm which is a discrete system state recursion:

\[ X_{N}=B_{1}X_{N-1}+B_{2}U_{N} \]
Petri net, enabling tokens are encumbered or reserved when a transition fires. After a specified delay time associated with the transition, tokens are deposited at the output places of the transition. Thus, a timed Petri net is used, according to the present invention, to provide an effective method for modeling the finite delay time associated with the completion of a primitive operation by a processor.

As discussed above, the first step in executing an algorithm according to the present invention is to decompose the algorithm and describe it using an algorithm directed graph. A simple example of an algorithm directed graph is illustrated in FIG. 1. This graph is characterized by nodes which represent the operation or function to be performed on a data packet or vector incident to that node. Directed arcs or "edges" of the graph represent the data labels of inputs to the primitive operations. Edges leaving a node represent the output of the primitive operations which are passed on to the next primitive operation in the algorithm.

In FIG. 1, node 10 represents supplying the Nth value of A. When this data is supplied, a token is indicated on edge 12. A constant value is supplied on edge 14 from node 15 to node 16. When tokens representing both UN and A are available, the primitive operation V = V + V adding the value of B = UN to A is provided and supplying the value XN to edges 22 and 24. While edge 22 supplies the value XN to an output node 26, edge 24 supplies the value XN to node 28 which performs the primitive operation V = V. Edge 30 supplies the constant B from node 31 to node 28 so that the value B = XN can be supplied from node 28 to node 20 on edge 32.

Although the algorithm directed graph is useful in describing the computational flow, it does not address the procedures a particular computing structure must manifest in order to perform the task indicated on the graph. Further, the graph is ambiguous with regard to issues of protocol, deadlock and resource assignment when processing is to be achieved in a multiple processor environment. Of particular importance is how the sequence of data flow and operations are managed in a multiple computational resources environment which emphasizes maximum concurrent processing. An unambiguous representation of control data flow is provided by a particular variety of Petri net termed a computational marked graph which permits data concerning control status and resource utilization to be monitored by the placement of "tokens" which may be tracked during execution of an algorithm. The development of the computational marked graph is accomplished by generating a node marked graph for each node in the algorithm directed graph.

In the preferred embodiment, the node marked graphs are generated after making certain assumptions about the system which will execute the algorithm. It is preferable to assume that the system comprises a plurality of functional units each of which includes processing capabilities, local memory for programmed storage and temporary input and output data containers. It is further assumed that a global data memory will be available to all functional units. Inputs associated with each edge in the algorithm directed graph are fixed data containers in the global data memory. In the preferred embodiment, transfer of data from a functional unit through the global data memory is postponed until all successor computational nodes have accessed the data. This requirement helps avoid processing time, but leads to the necessity of providing data status information in the computational marked graph. Finally, a primitive operation will not be assigned to a functional unit unless the functional unit is available and all data required by the primitive operation is available.

As a result of the requirement in the preferred embodiment that output data from a primitive operation be accessed by subsequent nodes prior to being stored in the global data memory, the node marked graph is developed with data status information being directed along an edge corresponding to each edge providing data between two nodes, but in a direction opposite to that of the data flow. In addition, one embodiment of the present invention breaks down each node of the algorithm directed graph into three nodes: reading, processing and writing. Such a three node marked graph is illustrated in FIG. 2, where node 40 is a reading node, node 42 is a processing node and node 44 is a writing node. Edge 46 provides a first input 11 and edge 48 provides a second input 12. Edge 50 provides data status corresponding to the first input 11 on edge 46, while edge 52 provides data status corresponding to the second input on edge 48. A data ready signal is supplied along edge 54 to the processing node 42 when inputs 11 and 12 are full as indicated by tokens on edges 46 and 48 (see FIG. 3B). When the process or computation are formed by processing node 42 is completed, a token is placed along edge 56. Data output edges 58 and 60 are directed outward from output node 44, while edges 62 and 64 provide data status information regarding the use of the data output on edges 58 and 60, respectively, by subsequent primitive operations. When the output buffers have been emptied, a token is placed on edge 66 to indicate that the process is not busy. This state is represented by the tokens 68a-68e illustrated in FIGS. 2 and 3A.

Eight possible states are illustrated in FIGS. 3A-3H. They respectively illustrate the idle state (FIG. 3A) which is also illustrated in FIG. 2; process ready - inputs enabled (FIG. 3B); data accepted - inputs consumed (FIG. 3C); performing processing (FIG. 3D); processing completed - outputs available (FIG. 3E); processing completed - inputs available (FIG. 3F); data consumed - output buffers full (FIG. 3G); and performing processing - output buffers full (FIG. 3H).

As discussed above, a computational marked graph is provided by replacing each of the nodes of the algorithm marked graph illustrated in FIG. 1 with a node marked graph like those illustrated in FIGS. 2 and 3A-3H. For the algorithm directed graph illustrated in FIG. 1, the corresponding computational marked graph is illustrated in FIG. 4. In other words, the computational marked graph of a particular algorithm is composed of interconnected node marked graphs by joining the various input edges to corresponding output edges of predecessor node marked graphs.

Preferably, the computational marked graph according to the present invention has several features. First, no more than one token is allowed in each place (edge) at a time. This is significant in that the occurrence of more than one token in a place would indicate overrunning data buffers or perhaps multiple allocation of a process. Second, both data flow and buffer full conditions are represented by directed arcs corresponding to the directed data flow of the original algorithm directed
necessary programs are stored in its memory and the 65 For a node in the computational graph with two 60
proper connections are provided. For example, for inputs, 65
testing purposes, a parallel processing system has been marked 70
graph corresponds to the graph illustrated in 75
constructed in which the algorithm loader is an IBM FIG. 104.

The functional units 105-108 and only if the number of tokens on each cycle is at least 35 follows. The assignment manager 200
may be differentiated by the functions which they are intended to perform. For example, the present invention 250
may contain a larger amount of memory if a large amount of data is being processed. Also, one or more of the functional units, such as func-

One example of a parallel processing system which initially activated functional unit is made available, the 400
algorithm loader assigns the primitive operation which uses the produced data to another functional unit. Preferably, only the primitive operations to be 450
performed by a functional unit and the data labels are assigned by the assignment manager 104. All data access 500
and processing operations are performed by the functional units 105-107. Data from the functional units 105-107 is stored in and supplied to the functional units 105-107 by the processor in the global memory 108. 550
As execution of the algorithm proceeds, the assignment manager monitors the completion of each transition 600
node by marking the appropriate edges or places in the computational marked graph stored in its memory with "tokens".

As illustrated in FIG. 5, the algorithm is executed in k functional units 105-107 by accessing data stored in the global data memory 108 via the data bus 112. Control and status information are passed between the functional units 105-107 and the assignment manager 104 via the token bus 110. The assignment manager 104 is primarily engaged in managing the buffer status of the inputs and outputs of each node. The operation of the assignment manager 104 and the type of data passed over the token bus 110 will be described below with reference to FIG. 1.

For a node in the computational graph with two inputs, I1 and I2, and two outputs 01 and 02, the node marked graph corresponds to the graph illustrated in FIG. 2. As noted above, the assignment manager 104

PC/XT and each of the units 104-108 include an INTEL 8088 processor, 32 kbytes of memory and the required number of input/output ports in an S-100 bus enclosure with a power supply.

In the test system, the units 104-108 and 102 have been connected by RS-232 cables transmitting at 9600 baud. However, the present invention is by no means limited to such a system, but could be operated on any parallel processor system which includes a device to carry out the functions of the assignment manager described below. For example, the present invention could be applied to a system in which the functions of the assignment manager are performed by one or more of the functional units. Similarly, while the global memory 108 is illustrated as a separate unit, in an actual system, the memory associated with each of the functional units 105-107 might contain a copy of the contents of the global memory with transmission between the units occurring at the time that the data is to be stored in the global memory. In an actual system, it is expected that the units 104-108 may each comprise a single printed circuit board or even a single integrated circuit. In addition, instead of being identical units, the units 104-108 may be assigned to functional unit 108, which might include additional input/output ports for receiving or transmitting data to or from the external data unit 114.
includes both processing capability and memory. The memory is used to store the following information for each node of the graph: node number; processing indicator; function name; and buffer status indicators for I1, I2, 01 and 02.

The data passed between the assignment manager 104 and the functional units 105-107 may be viewed as asynchronous handshaking. In one embodiment of a dataprocessing system to which the invention can be applied, the following messages are transmitted. These messages may be hard-wired or in the form of data packets.

To initiate the start of a computation, the assignment manager 104 first determines that node #1 is fireable by determining the status bits associated with the inputs of node #1 indicate that all input buffers are full (11-f, 12-f) and the process is not busy (NB) as illustrated in FIG. 2. The algorithm directed graph contains at least one node that has been recognized, the assignment manager 104 sends a READY message over the token bus 110 to all of the functional units 105-107. The READY message preferably includes the name of the function to be processed and the inputs to be used. The available functional units may respond with an ACCEPT message containing the functional unit identification number and the function name. The assignment manager 104 acknowledges one of the ACCEPT messages with an IDACK response indicating the identification number of the functional unit to be assigned the function, the function to be performed, and the node number of the graph. The assignment manager 104 then updates the computational marked graph in its memory to indicate that the node corresponding to that node number (#1) has been assigned to a functional unit, i.e., functional unit 105. As soon as the indicated functional unit 105 has retrieved the input data from the global data memory 108, it transmits an INPUT message indicating its identification number and the input buffers which have been used. The assignment manager then updates the input status bits of the corresponding node in the computational marked graph stored in its memory and responds with an INACK message to acknowledge receipt of the input message. When processing is completed, the functional unit executing the primitive operation transmits an output request message OUTREQ to the assignment manager 104 over the token bus 110. The output request message would include the functional unit number and the number of the node being processed. The assignment manager 104 responds with an output status message OUTSTAT indicating the identification number of the functional unit and the labels of the global data memory 108 to be used for the outputs 01 and 02. The functional unit 105 responds by transmitting the output data to the global data memory 108 via the data bus 112. When this process is complete, the functional unit 105 transmits an OUTPUT message to the assignment manager 104 via the token bus 110. The assignment manager 104 responds by updating the computational marked graph in its memory and transmitting an output acknowledge message OUTACK to the functional unit. A system such as that described provides the benefits of highly efficient concurrent processing of primitive operations in an algorithm while minimizing the efforts involved in programming and providing a large amount of fault tolerance, even the failure of a functional unit during execution of the algorithm. In addition, the number of functional units need not be known at the time that the algorithm is defined as a computational marked graph.

Many other features and advantages of the present invention are apparent from the detailed specification, and thus, it is not desired to limit the invention to the exact construction and operation illustrated and described. For example, while graph terminology has been used in describing the operation of the assignment manager, the present invention is not limited to a graph oriented system, but rather is applicable to any concurrent execution of an algorithm in which primitive operations are continuously assigned by functional units during execution of an algorithm by an assignment manager. Accordingly, suitable modifications and equivalents may be resorted to, all falling within the scope and spirit of the invention.

What is claimed is:

1. A method for concurrent execution of primitive operations in an algorithm by a dataprocessing system having a data-driven architecture comprising an assignment manager, a plurality of functional units connected to the assignment manager and a global data memory connected to the functional units, said method comprising the steps of:

(a) generating an algorithm directed graph containing nodes representing primitive operations and data flow arcs representing data flow in the algorithm;

(b) automatically providing definitions of the algorithm directed graph generated in step (a), wherein the algorithm directed graph contains at least one node for each primitive operation that has been defined as the data for that primitive operation becomes available, said assigning performed by the assignment manager; and

(c) assigning each primitive operation to an available function unit in dependence upon the definitions previously defined as the data for that primitive operation becomes available, said assigning performed by the assignment manager; and

(d) monitoring in the assignment manager completion of each primitive operation to determine data availability.

2. A method as recited in claim 1, wherein the algorithm directed graph contains at least one processing node, and wherein step (ii) comprises the step of defining input, processing and output transitions for each of the processing nodes in the algorithm directed graph.