A Data-Flow System for Decentralised Control

G. K. Egan
Department of Computer Science
University of Manchester
Oxford Road
Manchester
M139PL
England.

Background

Modern Control Theory has failed in its attempt to deal with Large Scale Multivariable control problems. It demands that all process state information be made available to a single central controller (centrality) and this has led to serious problems with communications, computational tractability and reliability [Sandell].

Decentralised Control Theory and inexpensive computational elements in the form of the microprocessor, both developed recently, hold the hope of extensive control decentralisation in Large Scale Multivariable control systems. This may eventually lead to decentralised computing systems with hundreds of computational elements.

The Problem

The models of conventional computing systems also assume centrality. However, they have limited practical value and some positive disadvantages [Backus].

Control engineers are faced with the unsatisfactory prospect of resting their decentralised control models on computing systems and models of dubious practicality. They urgently require practical computing systems which are a match for the highly concurrent environments of decentralised control.

Possible Solution

One possible computing model, with a sound mathematical basis, which seems well suited to decentralised control is Data-Flow [Karp, Adams etc.]. It is of a stimulus-response type; communication information is inherent in the model; it is asynchronous and so well suited to systems with loosely coupled processing elements; the concept of centrality is absent.

Experimental System

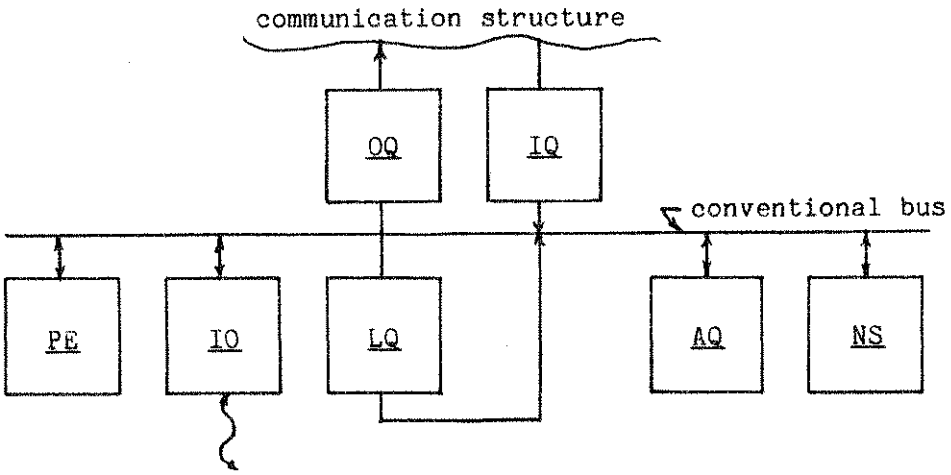
Computing systems for decentralised control will consist of a large number of asynchronously communicating modules co-operating on some overall control task. Communication paths between modules may be bit-serial, or at best byte-parallel, and the modules themselves of limited computational capability. It is important therefore not to overload communication paths and modules with excessive token tagging schemes and their supporting primitives. Furthermore concurrency in decentralised control is predominantly explicit. The more implicit forms of concurrency, which systems with large token context tags are designed to exploit [Arvind, Gurd], are less evident.

Minimal tagging does not severely limit us however as sub-graph sharing and even multiple recursion is still possible. The only tags in this architecture are those associated with token destination (node description location), mode (type and length of the token data field) and only when sharing a sub-graph, a copy number.

Given the above, the main requirement of the hardware is to maintain the strict queueing of tokens on arcs for any given sub-graph invocation. This is not as difficult as it may first appear as we know quite a lot about the pattern of arrival of tokens at any given node type.

System Modules

The structure of the modules (Fig. 1), while far from optimal, attempts to maximise flexibility in the initial experimental configuration; it also allows us to assess the practicality of evaluating data-flow graphs on conventional systems.



System Module
Figure 1

Each module is comprised of several sub-modules and, in the initial implementation, they are as follows:

PE (processing element) A shared controller and execution unit,

NS (node description store) A conventional store containing descriptions of graph nodes,

IQ (input queue) A hardware FIFO buffer through which the module receives tokens from other modules,

LQ (local queue) A software queue containing tokens generated by the module which are destined for the same module (similar to the agenda queue of Davis's DDM1)[Davis],

OQ (output queue) A hardware FIFO buffer through which the module transmits tokens to other modules,

AQ (arc queue) A linked list structure containing tokens which are queued on one arc of two-input-arc nodes.

The linked list structure in AQ is the means by which the queueing of tokens on arcs is maintained within modules and as such deserves some comment. Each two-input node description has a link into AQ,the first entry of which contains the following:

1) The input-point on which tokens are currently queued,

2) A pointer to the first token on the arc,

3) A pointer to the last token,

4) If the module supports sub-graph invocations, a copy number and a pointer to a similar entry for any other active invocation.

Module operation is fairly simple and proceeds as follows:

The PE scans LQ and IQ until a token is present.

The node description corresponding to the token which has just arrived is examined to see if it has one or two inputs.

If the node has one input, the PE evaluates the node-function and writes any resulting tokens to OQ or LQ.

If the node has two inputs, AQ is accessed via the node link for a matching token.

If the match is successful, the node-function is evaluated; otherwise the token is linked to the end of the appropriate arc queue.

The PE then returns to scan LQ and IQ.

The detailed behaviour of a module may vary depending on its primary function ie. input-output, computation etc.

Shared Sub-graphs

Many modules may not support shared sub-graphs eg. simple controllers at the periphery of a control system. However, for those modules which do support shared sub-graphs, the following mechanism is used to separate different sub-graph invocations. On entry and exit from a sub-graph a copy number is computed using the equations below:

entry newcopy = (oldcopy * maxoccurrence) + occurrence

exit newcopy = (oldcopy - occurrence) DIV maxoccurrence

It is important to note that the copy number is only appended to tokens actually involved in a shared sub-graph invocation. The copy numbers correspond to branches of an n-ary tree.

Don't-knows (?)

Most of the token types in the system are conventional for example real, integer, character, bit-string, end-stream. However some are less conventional such as node; this is used when loading graphs onto the system.

One of the more novel types is the mode ?. While it may be used in many applications eg. partial pattern matching , it's main use is to communicate information about exceptions occurring during the evaluation of a graph to the graph itself. In this system exceptions fall into two classes:

1) Faults in evaluation or attempted evaluation of node functions eg. function argument mode exceptions (inc. I/O), arithmetic exceptions, range exceptions (data windowing) etc. With this class of exception a ? token can be safely propagated to succeeding nodes. ? tokens propagated in this manner retain the original reason for exception and the destination at which that exception occurred. The ? token can not be used as a control token

on conditional path nodes (Pass-if-True, Pass-if-False, Switch).

2) Destination exceptions e.g. non-existent node description, inactive node input-point etc. Because no successor node exists for this class of exception, a reserved exception-node is defined in each module. A token of mode destination is sent to one input-point of the node and any ? arriving at the other input of the exception node is sent to that destination.

Input-output

Input and output node names are reserved and are associated with particular devices. The actions of input-output nodes are as follows:

1) Input A response destination, which remains valid until another arrives, is sent to one input-point; to the other is sent a token of any mode. Depending on the nature of the device, the associated input node will eventually respond with valid data or a ?. If no response destination has been specified, a ? is sent to the module's exception-node.

2) Output A response destination, as for input, is sent to one input-point and data to the other. The node responds with a copy of the original data or a ?. If no response destination is specified then a ? is sent to the module's exception-node only when the output action fails.

Storage-nodes

In adaptive controllers it is necessary, from time to time, to update "constants" in for example the difference equations which represent the digital compensator. While it is possible to retain information by circulating these "constants" it is, to say the least, not very efficient.

The approach taken here is to provide a storage node. One input-point of this node receives written tokens while the other, when receiving any token, causes a copy of the last written token to be transmitted. If no token has been written a ? is issued.

Because read and write operations are not synchronised, graphs using storage nodes may pass through short periods of "fuzzy determinacy" when write actions occur.

Status

Most of the system and process environment is being simulated on a large conventional system (MU5) [Ibbett]. Four modules have been constructed and are currently being commissioned. These are connected to MU5 and represent real modules in the overall system.

Research in progress includes:

1) Application-specific graphical languages.

2) Graph partitioning techniques to minimise inter-module communications using input-output nodes as clustering centres.

3) Applications including object recognition using a "laser tracker" [Ishii].

4) It is quite possible to configure modules which exploit concurrency in arc queueing, node execution and result transmission to a far greater degree than the experimental module structure. Such modules are being investigated.

References

[Adams] Adams, D.A., "A Model for Parallel Computations", in Hobbs (ed) Parallel Processor Systems, Technologies and Applications, Spartan Books, 1970, pp311-333.

[Arvind] Arvind & Gostelow K.P., "A Computer Capable of Exchanging Processors for Time", Information Processing 77, North Holland, 1977, pp849-853.

[Backus] Backus, J., "Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs", CACM Vol. 21 No. 8, Aug. 1978.

[Davis] Davis, A.L., "Architecture of DDM1: A recursively Structured Data Driven Machine", Technical Report, Dept. of Computer Science, University of Utah, 1977.

[Gurd] Gurd, J.R., Watson I. & Glauert J.R.W., "A Multilayered Data Flow Computer Architecture", Draft document, Dept. of Computer Science, University of Manchester, Jan. 1978.

[Ibbett] Ibbett, R.N. & Capon P.C., "The Development of the MU5 Computer System", CACM Vol. 21 No. 1, Jan. 1978.

[Ishii] Ishii, M. & Nagata T., "Feature Extraction of Three-Dimensional Objects and Visual Processing in a Hand-eye System", Pattern Recognition, Vol. 88, p229.

[Karp] Karp, R.M. & Miller R.E., "Properties of a Model for Parallel Computations: Determinacy, Termination and Queueing", SIAM J. Applied Mathematics, Vol. 11 No. 6, Nov. 1966, pp1390-1411.

[Sandell] Sandell, N.R. et al., "Survey of Decentralised Control Methods for Large Scale Systems", IEEE Transactions on Automatic Control, Vol. AC-23 No. 2, Apr. 1978, pp108-128.