[illegible] DCS [illegible]

SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

DISTRIBUTED COMPUTING NOTE 522

VISITS

issued by
D A Duce

Notes on a Visit to Dr J Gurd et al
University of Manchester 16 November 1981

24 November 1981

DISTRIBUTION:	R W Witty
	F Chambers (Logica Ltd)
	D A Duce
	Miss G P Jones
	Investigators/Gurd and Watson

John Gurd gave an overview of the project and the data flow concept.

BACKGROUND

The project started with unfunded work (simulation) between 1976 and 1978. The construction of a simple ring prototype machine has been supported with DCS funds from 1978-81. A rolling grant has just been awarded (1981-85) for the construction of a multiring system. In addition a third grant (1981-84) has been awarded for the construction of system software for the prototype (a cross compiler in the first instance).

Algorithm design is being considered under the rolling grant. Staff resources to 83 are:

1 RA cross compiler
1 RA algorithm work
1 RA hardware for multiring system

There are in addition several PhD students working on related topics.

OVERVIEW

In the mid 70's such techniques as pipelines, vector pipelines and multiple functional units were being used to introduce parallelism into hardware. However conventional programming techniques were being used in support of these developments and there were concerns as to how far that could go. Providing the parallelism in the problem was of the requisite sort (eg vectors appropriate length) such techniques worked well, but they could only work for regular parallelism.

Consider the program fragment:

```
c =
a = b*c+d*e
c = a+x
y = x-c
```

Dataflow analysis turns this into a 2-D graph:

$$\begin{tikzcd}
b \arrow[dr] & & c \arrow[dl] & & d \arrow[dr] & & e \arrow[dl] \\
& * \arrow[dr] & & & & * \arrow[dlll] & \\
& & + \arrow[dr, "a"] & & & & \\
& & & + \arrow[dr, "c"] & & x \arrow[ll] \arrow[dl] & \\
& & & & - \arrow[d] & & \\
& & & & y & &
\end{tikzcd}$$

Enhancements permitting cycles, loops, conditionals and functions are possible.

An execution model for such a graph is the following: data is carried by tokens which are pulled towards the foot of the graph. All data values have an associated tag - the destination.

Operands are consumed by operators, thus if more than one copy of a data value is required (eg x in the graph above) these have to be generated explicitly by copy operators.

Data structures can be represented as a bunch of tokens flowing down an arc together. There are many possibilities. One is that the arcs may be FIFO's and components of a structure are identified by their positions in the queue. This technique is difficult to implement in hardware (unbounded FIFO's). An alternative approach is to colour tokens - a colour being an extension of the tag field.

A further problem is to distinguish tokens generated on different cycles of a loop or levels of recursion. Recursion can require a FILO mechanism (ie consume in reverse order to generation). Token colouring covers this case.

The Manchester machine uses the token colouring model. 96 bits are allowed for data, colour and destination. The largest value that can be carried is a 32 bit real. This wastes a lot of store but is acceptable for a prototype machine.

To connect this execution model into hardware, three things are required:

(i) a mechanism to carry tokens into heads of arcs
(ii) a matching process to match pairs of tokens
(iii) processing

The execution cycle is then:

$$\begin{tikzcd}
\text{match} \arrow[r] & \text{pick up destination} \arrow[r] & \text{process} \arrow[ll, bend left]
\end{tikzcd}$$

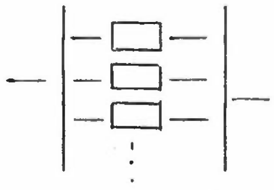
This is realised in the following skeleton architecture:

$$\begin{tikzcd}
 & \text{matching unit} \arrow[d, bend left] \\
\text{progressing unit} \arrow[ur, bend left] & \text{graph store} \arrow[l]
\end{tikzcd}$$

The matching store looks for tokens with the same colour and destinations differing only in the arc field (left or right). The matching visit has an associated pool of tokens - if the partner for the token just received has already arrived, the pair are dispatched to the graph store, otherwise the token goes into the pool to await the arrival of its partner. The design of the matching store is a critical part of the architecture.

The graph store contains one entry per node. The executable package from the graph store may have up to two tokens and two destination fields (eg copy operation).

The processing unit consists of a number of processors:



The package may execute on any free processor.

There are two units missing from the simple ring above, one is the communications switch between processing unit and matching unit, which allows tokens to be injected into and removed from the ring, the other is a FIFO token queue between the switch and matching unit, which allows parallelism to build up without the entire system becoming deadlocked.

The matching unit is an associative store implemented as a hash table. There are 8 memory banks in parallel, when the hash address has been formed, the 8 banks are searched in turn for a matching value.

If a match is not found, the token will be inserted at the hashed address in the next free bank. When all 8 positions corresponding to that address are full, the token is dispatched to be dealt with by the overflow unit (LSI-11) "at its leisure". The overflow unit can safely be an order of magnitude slower than the main unit.

There is a counter associated with every line in the hash table which gives the number of values for that line in the overflow unit.

One can envisage many different ways of dealing with overflow and migration of tokens from the overflow unit to the main hash table when lines become free. Investigation of the performance of different strategies is a question for research.

The inter unit interfaces use identical hardware which means that single visits can be emulated by a host, tested against a host etc.

The system now contains at least one example of each type of board required. 3 PE's are now working, it is estimated that somewhere between 15 and 20 will be necessary to keep the machine running at maximum speed.

Additional memory boards are required for the token queue, node store and matching unit.

Each unit around the ring is synchronous with its own clock. Inter-unit links are asynchronous with full handshaking resynchronisation between units.

The beat time of the ring is 200 ns. The processing elements are micro coded; there is one writable control store and one clock (200 ns cycle time) for all pe's. Instructions take an average of 5-6 microsecs. The processing unit should on average deliver results every 200 ns, the matching store and graph store every 300 ns, (wider highways).

The host interface is only capable of producing tokens every 170 microsecs.

The processing elements are running at full speed, but other units have been detuned to just keep the three pe's currently operational, occupied. The remaining pe's should be completed by the end of January. A redesign of the matching unit controller board may be necessary to get this unit up to specification, but this is not regarded as disastrous.

All tokens flowing through the communications switch can be monitored. By post processing this data (collected on 11/02 magnetic tape deck) statistics can be gathered, eg how long tokens stay in matching unit.

John is making a case for a PERQ - statistics are presented in graphical form; the ability to browse through the data would be a valuable asset to the evaluation programme. They also want a means of linking the host, overflow store, PERQ and 11/23, ie they want to borrow a Cambridge Ring also.

SOFTWARE

There are three strands to consider:

1 Fortran/Pascal etc.
2 (Graphical coding) simple assignment languages.
3 Zero assignment languages.

1 They are trying to get hold of Kuck's Parafrase system which could be used to generate dataflow machine code from Fortran.

2 Languages such as LAU, Id and Val (Dennis MIT) are being considered. John Glauert is going to the States next summer with a brief to pick up the end-user consensus on an acceptable language for programming dataflow machines.

They are keen to pick up Val and Id, but both pose problems. The Val compiler is written in CLU, the Id compiler in MACLISP.

They have been offered the Id compiler code and asked if they could get time on the Edinburgh DEC 10s (say 50-100 AU's). DAD to investigate.

3 LISP, SASL etc. Reduction systems - code rewriting. There is a very close link between single and zero assignment languages.

DEMONSTRATION

We were shown the dataflow machine in operation calculating factorial 5 by Gurd's double factorial recursive algorithm. This showed an approximately line-on speed up using 1, 2 and 3 processors (processors can be switched in and out independently).

GENERAL POINTS

1 John and Ian think it is the basic ideas in the machine that are useful, rather than the realisation in the particular technology they have chosen. They would like to see other machines of a similar nature to see how they compare.

2 John has received several inquiries about using the machine. It was agreed such requests should be channelled through SERC.

3 John asked about ARPAnet access. John has been given form and will complete in due course. DAD to look into how best to complete the form.

4 John would like to be kept in touch with what is happening in Japan.

5 They have had an offer of collaboration from a computer company and requested clarification of the general principles.

6 They would like to know what Roger Newey's view is now. DAD to investigate.

7 They think they will be ready for the formal review of the evaluation programme by July 1982. If a PERQ is to have a part in this programme they would need it before the end of March at the latest.

8 A man from DEC who is interested in PCB routing algorithms is visiting Machester in December for 10 days to try his algorithms on the machine.

9 Can Ian get an earmarked studentship for next year which would not reduce the departmental quota in some sense.