UDK 681.519.7

Peter Kolbezen
Institut »Jožef Stefan«

A communication mechanism based on the exchange of the distributed topology information is discussed. A particular reconfiguration technique is treated to handle the exchange of topology information and thus to maintain the necessary routing tables. This mechanism handles the reconfiguration dynamically. Inmos Transputer concepts are introduced and applicated on the two-dimensional square interconnection structure.

"Rekonfigurabilni" večprocesorski sistemi. Prispevek obravnava komunikacijski mehanizem, ki je zasnovan na spremembah topologije komunikacijskih poti med mikroprocesorskimi enotami sistema. Namenjen je posebni t.im. "rekonfigurabilni" tehniki, ki obravnava spremembo informacije o topologiji in na ta način vzdržuje potrebne razpredelnice povezav medprocesorskih komunikacij. V predlogu so vpeljani koncepti Inmosovega transputerja, kot vozliščnega procesorja v povezovalni dvodimenzionalni kvadratni strukturi.

1. INTRODUCTION

Highly parallel computing structures promise to be a major application area for the milion-transistor chips that will be possible in just a few years. Such computing systems have structural properties that are suitable for VLSI implementation. The key attributes of VLSI computing structures are

- simplicity and regularity
- concurrrency and communication
- computation intensive VLSI.

The choice of an appropriate architecture for any electronic system is very closely related to the implementation technology. This is especially true in VLSI. The supervisory overhead incurred in general-purpose supercomputers often makes them too slow and expensive for real-time and signal and image processing. Progress in VLSI technology has lowered implementation costs for large array processors to an acceptable level. Algorithmically specialized processors often use different interconnection structures. The matching of the structure to the right algorithm has a fundamental influence on performance and cost effectiveness.

An alternative to the design of a globally synchronous array is to achieve a self-timed system through the use of asynchronous handshaking mechanisms established between neighboring processing elements. These self-timed implementations are commonly referred to as **wavefront arrays** /8,12,21,23,25/. The wavefront array combines the systolic pipelining principle with the dataflow computing concept. A wavefront array processor may be used either as an attached processor interfacing with a compatible host machine or as a stand-alone processor equipped with a global control processor. Such system consists of the processor array(s), interconnection network(s), a host computer and interface unit.

Exploitation of the dataflow principle makes the extractions of parallelism and programming for wavefront arrays relatively simpler.

A family dynamically interconnected or reconfigurable VLSI processor arrays allow an array to support a large class of algorithms. Such structures usually involve significant hardware overhead. Reconfigurability of array structures based on switching lattices has been proven to be useful for solving problems related to fault tolerance. Fault tolerance is an important concern in systolic and wavefront arrays because of the large number of processor elements they may have.

The paper concludes with suggestions as to how a recofigurable sistem may be designed with constructing a wavefront array with the family of Inmos Transputers /20/. Transputer chip is an Occam-language-based design that provides hardware support for both concurrent computation and communication. It adopts the now-popular RISC architecture. Its features make it a powerful building block for constructing concurrent processing networks. The transputer's links are the hardware representation

of the channels for process communication. There is an intimate relationship between transputer channel links and the communications protocol envisaged for wavefront arrays.

2. ADAPTIVE SYSTEMS

One of the major objectives of the adaptive systems is to be rearrange the configurations to match the needs of different applications. A such flexible **Reconfigurable Multi-microprocessor Systems** (RMmS) should include provisions for insertions (extensions) and deletions (reductions).

In general, a node in a system becomes aware of the entire configuration either by a central network control (CNC) or as result of distributed exchange of the configuration (topology) information. The CNC approach can be utilized to implement a fixed routing mechanism. The necessary routig tables, which are computed using the system topology information, are distributed to the participating nodes, once and for all. For a RMmSystems which does not have an CNC, it is necessary to compute routing tables externally and to provide each node with a copy of the full configuration information. Given the network topology, the shortest distance algorithems, such as Dijkstra's algorithm, can be used to compute all the shortest paths between the local node and the rest of the system /5,6/.

The provision of a distributed reconfiguration mechanism eliminates the need for a central network control and a copy of the full configuration information at each node. Instead, each node is responsible for maintaining partial routing information so as to be able to communicate to its neighbouring nodes.

The handling of configurational changes /5,7, 12/ in multi-computer/processor systems is a necessary part of their design. It is required for realiability, modularity, and extensibility. It is the dynamical reconfiguration which is important. The routing mechanisms, such as

- random
- flooding
- ideal observer, and
- adaptive

can respond dynamically to the changes. The frst two techniques can utilize any live link and node. A node is not necessarily aware of the full system topology. The third technique needs a central network control which ideally is assumed to watch the entire system and is responsible for incorporating traffic and topology changes. The last technique, i.e. adaptive routing, is basically a flow control mechanism which can also respond to the topology changes in terms of some delay function.

In this paper, a communication mechanism based on the exchange of the distributed topology information is discussed. A particular reconfiguration technique is proposed to handle the exchange of topology information and thus to maintain the necessary routing tables. A topology message is sent only if there is an indication of a change in the configuration. This mechanism handles the reconfiguration dynamically. The link, node, or link and node failures or unexpected occupation and the reverse changes, i.e. new or free link, new or free node, or new or free link and node cases, are made known to every live node in the system in a finite time.

The treatmented reconfigurations technique in first part of the first work on this topic is discussed in connections with uniform and dense networks. It is based on Baran's hot-potato routine /2/. A correctness proof of a similar algorithm has also been presented by Tajibnapis /1/.

3. INITIALIZATION

The discussed reconfiguration algorithm is basically an adaptive algorithm. It is intended, however, to be controlled by the configurational changes rather than traffic changes. It will initialize the system by setting the initial configuration and then adapt the system, dinamically, to further possible changes in topology. The changes occur either as a result of failures or unexpected occupation (link, node link, and node) or as a result of new links, nodes, or links and nodes joining the system. The topology message format is as shown as:

header	destination	source	distance	crc

Given a network of **N** nodes, it first undergoes an initialization phase whereby each node detects the adjacent operable links and thus exchanges this information with neighbouring nodes. The information is passed in the form of standard format units (topology messages). The new neighbour receives the full account of the topology information available at the detecting node, in the form of one topology message per accessible node. Note that the links are assumed to be bi-directional, i.e. if a link **(a,b)** is live so is the link **(b,a)**. In practice this is not always the case.

The topology message contains two fields relevant to reconfiguration: identification of destination node **i** and the shortest distance between the sending node **s** and node **i**. A node generates, and may modify, a shortest distance table as shown in Fig. 1.

Destination	0	1	2	...	k	...	n-1
Distance	D(0)	d(1)	d(2)	...	d(k)	...	d(n-1)

Fig. 1 Shortest distance table at node k

The entries are indexed by the node identification and the entries themselves are the shortest distance between the corresponing destination node and the local node.

4. HANDLING INSERTION OF A NEW NODE

Now, suppose that node 1 of Fig. 2 detects a new neighbour, say node 17 in the same figure, and the rest of the system is uniform with the configuration known and node 17 has no ties with any other node in the system, except node 1. This is a major change in the network. The topology message exchange propagates until the configurational change has been detected at every node in the system. A receiving node

needs to send topology messages to its neighbouring nodes only if the topology message received has changed the previous topology information, i.e. the shortest distance table.

This process can best be visualized by a top-down tree, as shown in Fig. 3.

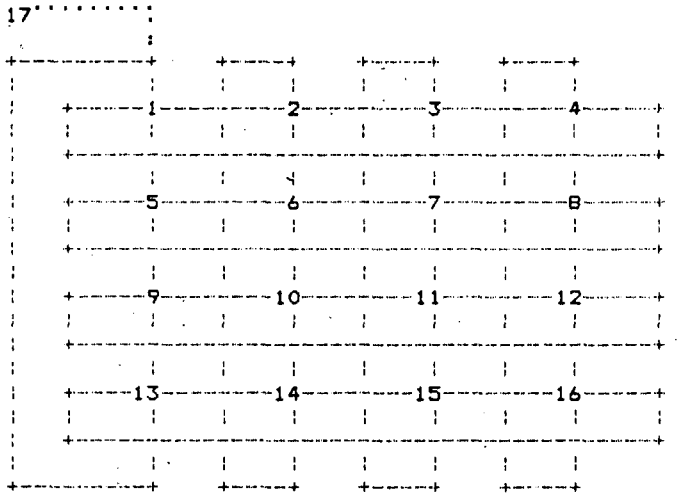

Fig. 2 Node insertion

The tree is extracted from the network shown in Fig. 2. The messages on the topological changes originated at node 1 stop at the terminal nodes when the configuration information, i.e. the shortest distance tabele, has reached settlement. Note that the branches represent the communication links and the vertices represent the nodes as numbered in Fig. 3.

A node is crossed if it has already been traversed once by an equivalent topology message. Two topology messages are considered equivalent if the destination fild, the distance field, and the sending node are the same. For this example a node is traversed at least once for the configurational change to be transparent to the system since node 17 has no ties with the rest of the network. However, the node may have to be traversed more than once in cases where the network is loaded with other traffic which may cause delays on certain paths. The topology messages that then take longer paths can reach a node before those taking sorter paths. The distance and the routing tables alter every time a topology change on a shorter path arrives. Every change is then fanned through the system.

Starting from the ROOT node 1, there are five levels of transmission. The system, with the exception of node 17, is expected to settle in five main periods. For node 17, however, (n-1) main periods are required to receive the complete topology information from the ROOT, where n is the number of nodes in the original network. This is because the ROOT needs to send the shortest path vector to node 17, where each path is transmitted in the form of a topology change message.

In practice, a new link does not always belong to a first-time node joining the system. The node might already have been taking part in the system. On the other hand, more than one link can become live at the same time. This occurs if one or more nodes join the system all at once, with all the adjacent links coming up simultaneously.

5. HANDLING FAILED LINKS

Link failures or occupationes handled differently. Obviously a failed or occupated link cannot take part in a transmission. Upon detection of a such link out of m+1 links, the prepared topology message only needs to be broadcast over the remaining m links. From then on, the procedure for handling a topology message is the same as for the live link detection case. Now, for clarity suppose that the failure/occupation of a particular link disconnects the adjacent node from the rest of the system, which is the reverse of the node insertion case discussed in the previous example. The failure/occupation of the link (1,17) causes the system in Fig. 2 to settle in five main periods. This is because the system requires five levels of transmission (see Fig. 3).

6. TRANSPUTER NODES

The transputer /13,15,20/ is a comporatively new hardware concept. The transputer products are aimed at highly parallel concurrent computing applications. The range of these products from Inmos offer system designers high band-

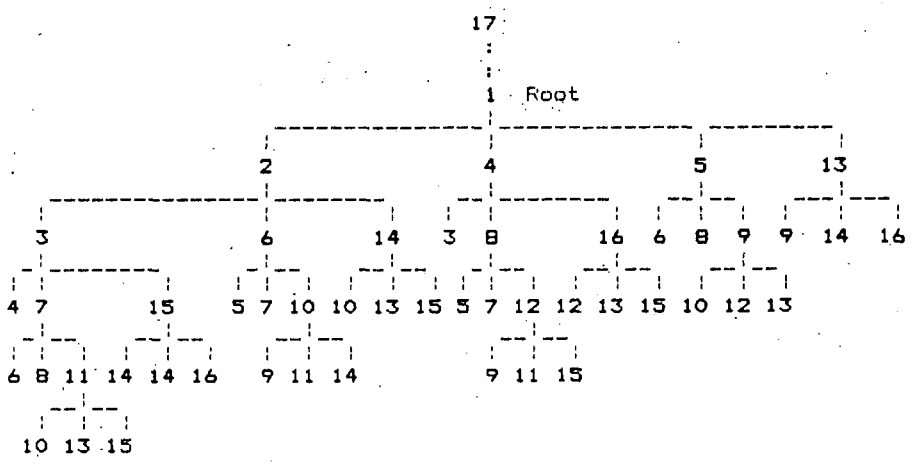

Fig. 3 Topology message exchange tree.

width interprocessor communications without a shared memory bus. It is a programmable VLSI device with communications links for point-to-point connection to other transputers. The software concept upon which transputer interprocess communication is based was originally propounded by C.A.R. Hoare /3/. All transputer range products share same basic common logical properties. These are:

- the ability of the processor chip to sopport external memory, but with the recommendation that processors should not share memory;
- the provision of high bandwidth serial links for interprocessor communication;
- hardware support for on-chip simulated concurrency, and for multi-processor parallel computing;
- low level software development in a high level structured notation.

A link between two transputers provides a pair of "occam" channels, one in each direction. A link between two transputers implemented by connecting a link interface on one transputer to a link interface on the other transputer by two one-directional signal lines. Each signal line carries data and control information.

Each message is transmitted as a sequence of single byte communication, requiring only the presence of a single byte buffer in the receiving transputer to ensure that no information is lost. After transmitting a data byte, the sender waits until an acknowledge is received. The acknowledge signifies both that a process was able to receive the acknowledge byte, and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

Data bytes and acknowledges are multiplexed down each signal line. An acknowledge is transmitted as soon as reception of a data byte starts (f there is room to buffer another one). Consequently transmission may be continuous, with no delays between data bytes.

Transputer also support program modularity with the possibility of late decisions about which part of the network particular software should reside and a degree of run time choise about which processor should be used for running particular tasks. There is nothing in the transputer architecture and organization which contradicts the idea of dynamic reconfiguration of the network. But dynamic reconfiguration of the transputer network is not supported at present by Inmos except that they provide a cross-bar swich with some low-lewel software support. The reconfiguration is needed in part because the hardware for transputers supports only a finite (and normaly very small) number of links from each node.

For the configuration above on the Fig. 2 which use all four links of every transputer in a array node may be used the B003 Inmos transputer board. There is no provision for booting the code which on the B003 must be done via a transputer link. The problem has been avoided so far since it destroys the symmetry of the network and makes the program unnecessarily complicated for these examples. It is shows one way that the problem may be solved for a 16 node 4 x 4 two-dimensional array on the Fig. 2. An extra transputer, which may be on a B001 or B002 board, is used as the boot node which can connected via a UART to a host machine. This processor is node 17 on the Fig. 2 and it may be include in the loop of nodes 1,5,9 and 13.

7. IMPLEMENTING THE RECONFIGURATION ALGORITHM

A network of needs to be reconfigurated if one of the following topological changes occurs:

1. a new link is introduced (or a failed / occupated link is repaired or again free and returned to the system);
2. a new node is joined to the system (or a failed node is repaired or again free and reinserted);
3. a link fails
4. a node fails.

It is important that topological change of a permanent nature is made known to the entire network in a finite time /5/. This is where the reconfiguration mechanism comes in. It controls the flow of topological information both distributedly and dynamically. On the other hand, temporary changes such as short time link, node, or link and node failures need not be fanned through the system. Instead a temporary blockage on the failed unit, or rerrouting at the adjacent nodes, can be more advantageous until the failure state is removed. This is because the global reconfiguration mechanisms use up an important fraction of the available communication capacity of the network which would otherwise be utilized for the actual information transmission. For permanent failures, however, the reconfiguration mechanism pays off with higher reliability, adaptability, and extensibility.

In a real life system, permanent and temporary configuration changes need to be distiguished before taking any action. This can be handled by employing a re-try and time-out mechanism. Before declaring a failure as permanent the detecting node waits for a finite duration of time during which the failure is declared temporary and the necessary precautions are taken accordingly. This mechanism can also be used for the new links or nodes joining the system. However, a live link or a live node detection mechanism can be incorporated with the type of joining, i.e. whether it is a permanent attachment or a temporary one. Reconfiguration is considered only for the practical case of permanent topological changes.

8. DATA BASE OF THE RECONFIGURATION ALGORITHM

The reconfigurations alghorithm operates basically on two tables. The distance table (DT) which records the distance (path length) of each node in the system from the host node via each of the neighbouring nodes; the routing table (RT) which records the shortest paths only. For each destination a maximum of m paths can be identified, where m is the number of neighbours. The DT is an n by m table, where n is network size.

Figure 4 shows the DT a node 1 of the 16-node network given in Fig. 2. In a regularly connected homogenous network all the distance tables are of equal size. An entry **d(k,i,j)** is the distance of path (k,i) via the neighbour **X(j)**, where i=1,2,...,n and j=1,2,...,m. The nodes that are not accessible from node k have a correspoding entry of infinity in the table. For example, node 1 appears inaccessible from itself via any of its neighbours.

The size of routing table (RT) depends upon the

Destination	Via neighbouring nodes			
	2	4	5	13
1	infinity	infinity	infinity	infinity
2	1	3	3	3
3	2	2	4	4
		. . .		
16	4	2	4	2

Fig. 4 Distance table for node 1 of a 16-node network

specific routing mechanism being employd. In fact, RT is arranged using the distance table. An entry r(i,k) or RT indicates the neighbour node, say X(j), via which the node i is at a minimum distance from the host node k, i.e.

$$d_m(k,i,j) = \min_{1 \leq m} (d(k,i,j)).$$

The entries of RT change only if the minimum of the corresponding DT row changes. It would save processing time if a shortest distance (SD) table were used alongside the distance table. We have

$$SD(i) = d_m(k,i,j).$$

Figure 5 gives the routing table and the shortest distance table given in Fig. 4. Note that r(1,1) = 0 and SD(1) is infinity which means that node 1 cannot send a message to itself by means of routing tables. For applications where the communication protocols maintain information flow between the processes rather than the processors (nodes) independent of where they are residing, we can have SD(k) noninfinity, where r(k,k) = 0.

The reconfiguration algorithm operates on those entries which are related to the topological changes. For example, if a topological change concerns node 1, only the ith level of distance, routing, and shortest distance tables need to be referred to after detection. An exception is the detection of the change which requires either the shortest distance table to be sent to the new neighbour or the column of the distance table corresponding to the failed link to be updated.

Destination	1	2	3	. . .	15	16
Next node	0	2	2	. . .	4	13

(a) Routing table (RT)

Shortest distance	infinity	1	2	. . .	3	2

(b) Shortest distance table (SD)

Fig. 5 Routing and shortest tables at node 1 of a 16-node network

9. THE RECONFIGURATION ALGHORITHM

The reconfiguration algorithm is structured into three basic subalgorithms. Algorithm 1 handles the link and/or node failures (or occupation). A node failure or occupation are interpreted as the the simultaneous failures (or occupation) of the links. Algorithms 2 handles a new link and/or node coming up. A node is declared new/free if all the links connected to it become alive simultaneously. Algorithm 3 handles the topology messages which indicate the configurational changes. These algorithms are given in detail down. A reconfiguration protocol is complete only with a specific routing mechanism, queue management algorithms, and input-output handling routines at the lower lever of communication.

Figure 6 showes a system flow diagram of the reconfiguration mechanism excluding the flow level communication protocol routines. A more detailed description of three algorithm has been written by Bozyigit. Some simulation results using the reconfiguration algorithms are reported by Bozyigit and Faker /6/.

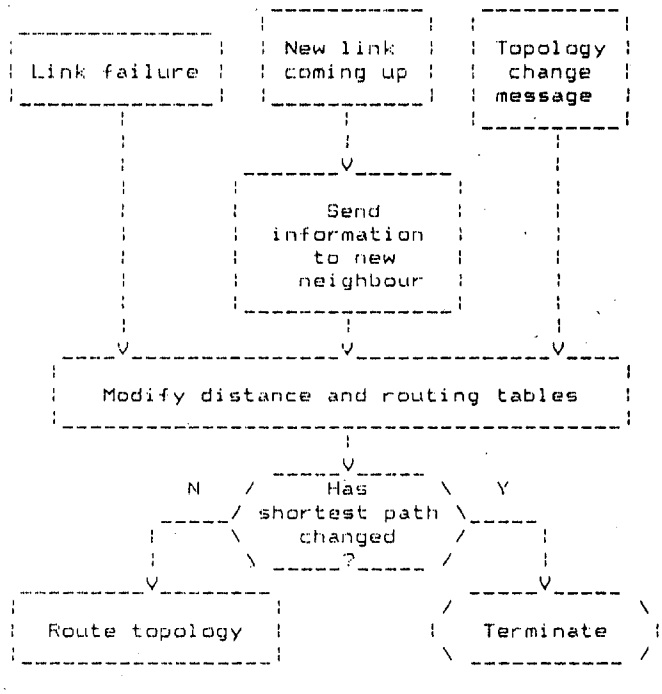

Fig. 6 Organization of reconfiguration algorithm

10. DESCRIBTION OF THE RECONFIGURATION ALGORITHMS

A. Nomenclature for the reconfiguration algorithms

a = source node
n = destination node
c = node adjacent (neighbour) to node a

d^a_{bj} = distance between a and b via the jth neighbour of a (the superscript a is omitted wherever it is obvious)

r^a_b = the first neighbour on shortest path between a and b indicated by a neighbour of a on that path

TM_b^a = topology message compiled at **a** to be sent to **b**

c <- TM_b^a = send TM_b^a to c

V_b^a = shortest distance between **a** and c

W_c^a = ith neighbour of **a**

X_i^a = ith neighbour of **a**

t = (a temporary variable)

B. Algorithms 1: Handles a new link coming up at node a

1. (a new link detected at the local node **a**)
   Link (a,c) becomes live;

2. (update distance and routing tables)

   d_{cj}^a : = 1 where jth neighbour = c,

   V_c^a : = 1,

   r_c^a : = c;

3. (preparate topology message)
   (set distance topology field of topology message)

   dist (TM_c^a) : = 1,

   (set destination field of topology message)

   dest (TM_c^a) : = c;

4. (send topology message to neighbouring nodes)

   X_j^a <- TM_c^a for j = 1,2,...,m and $X_j^a \neq c$;

5. (send available topology information to new neighbour)
   (form topology message)

   dist (TM_i^a) : = V_i^a,

   dest (TM_i^a) : = i for i = 1,2,...,n.

   (send topology change messages to new neighbour)

   TM_i^c : = TM_i^a.

6. Exit.

B. Algorithm 2: Handles link failures at node a

1. (a link failure or occupation detected at node a)
   link (a,c) failed or occupated;

2. (update the distance table)
   (save distance vector correspoding to failed/occupated link)

   t_i : = $d_{i,j}^a$,

   (set vector $d_{i,j}^a$ to infinity)

   $d_{i,j}^a$: = infinity where X_j^a = c, and

   i = 1,2,3,...,n;
   j = index of the failed or occupated link

3. (update routing tables and prepare topology messages)

   for i : = from 1 to n with step 1 do

   if $V_i^a = t_i$ then

   V_i^a : = min$|d_{i,j}^a|$, j ∈ m

   r_i^a : = X_j^a where $V_j^a = d_{i,j}^a$,

   TM_i^a : = V_j^a,

   (send topology message).

   TM_i^Y : = TM_i^a (where Y = X_j^a) for

   j = 1,2,...,m, except for

   X_i^a = c,

   fi,
   od;

4. Exit

C. Algorithem 3: Handles arrived topology message at node a

1. (detect topology message at local node a)

   TM_b^c arrives at a;

2. (update distance and routing tables)
   (assign new distance for (a,b) path)

   $d_{b,j}^a$: = $TM_b^c + W_c^a$,

   (find new min distance (a,b) path)

   t_b : = min $|d_{b,j}^a|$, j ∈ m

   (assign the shortest path)

   if t_b not = V_b^a then

   r_b^a : = X_b^a where $t_b = d_{b,j}^a$

   V_b^a : = t_b

   (prepare topology messages)

   dist (TM_b^a) : = V_b^a,

   dest (TM_b^a) : = b^a,

   (send topology message to neighbours)

   X_j^a <- TM_b^a for all j ∈ m

   fi;

3. Exit

E. Algorithm 4: Round-robin routing algorithm

1. (input a message at a) TM_b^c arrives

2. (find the shortest path (c,b))

   if c = (m-1) then
   c : = 0
   fi,

   (next node)

   c : = r_b^a;

3. (message joins output queue)

   OQ_c : = TM_b^a;

4. Exit.

11. CONCLUSION

Algorithmically specialized processors often use different interconnection structures of reconfigurable wavefront array processors /16,25/. There are a number of traditional algorithms which set up data structures in the first phase of processing, then applay for instance matrix arithmetic techniques to process the data, and in a file phase access the data using a different data paths. A trivial example is the hardware for a shift register with parallel in/serial out or parallel and serial in and out. The **mesh** is used for dynamic programming and the **heksagonally connected mesh** for L-U decomposition /12 /. The **tours** is used for transitive closure. The **binary tree** is used for sorting and the **double-rooted tree** is used for searching.

Each B003 board from the Inmos family has four transputers connected in as ring. There are therefore two links per transputer which can be connected exterally. In this manner there are a wide variety of networks that can be configured using only B003 boards:

- In the example of the shift register a ring of n transputers can be implemented with n/4 B003 boards. The transputers are connected serially as the process "node" together with four channels (two input, two output).

- A two dimensional array of 64 transputers (8 x 8) can be implemented with sixteen B003 boards. Four arrays of n channels (for a structure n x n B003 boards) are connected (left to right, right to left, top to bottom and bottom to top) so that each node can send or receive data to or from any of its four neighbours. Eight placed channels for n=4 (four input, four output) are passed to the process "node".

- A folded binary structure of size 4 x 16 can be implemented with 16 B003 boards too. The boards are connected externally. Four arrays of channels of dimension n (where the network is of size 2×2^n) are connected in such a manner: adj.left, adj.right, diag.left and diag.right and so on where "adj is a mnemonic for adjacent conection and "diag" for a "diag" for a diagonal connection. And also in this example eight placed channels (for input, four output) are passed to the process "node".

- A cube connected cycle of size 4 x 16 can be implemented with 16 B003 boards. Each row is configured on a single B003 board. Three arrays of cannels are defined of dimension n (where the network is of size 2×2^n): clockwise.rote, anti.clockwise.rote and cross.route, where "clockwise" and "anti.clockwise" represent channels connected to nodes in the same row, and "cross" represents a channel connected to a different row. This cube connected cycle is topographically equivalent to a four-dimensional hipercube with four nodes at each "corner". Six placed channels (three input, three output) are passed to proces "nodes". A mapping function is not necessary for this configuration since each transputer has the same link address for its six channels.

Ideally it would be desirable to allocate each process $p_j \in P$ (where P is collection of processes p_j) to exactly one transputer and to allocate each channel $c_{jk} PQ\%C$ (where C is the collection of channels c_{jk}) to one intertransputer link between transputers t_i and t_k - given that the matching pair of channels c_{ik} and c_{kj} may be allocated to the one link l_{jk}. Ideally if there are multiple channels between processes each should be allocated to different links. However there are practical limits to the number of links available to each transputer and there are possible electrical signal and wiring problems with geographically remote links in a large network. It some cases it is necesary to forward messages passively through some transputers at nodes in a network because there are not enough links on each device to provide all the direct communication that is required. There are some examples where such messages forwarding is a heavy overhead and it may be desirable to avoid it by dynamic hardware reconfiguration between the different phases of the program execution. Then the performance of the system may be improved at the expense of more complex hardware.

12. REFERENCES

/1/ A correctness proof of a topology information maintenance protocol for distributed computer network. W.B.Tajibnapis, Comm.ACM, July 1977.

/2/ On distributed communication networks. P.Baran, IEEE Trans. on Communication Systems, Vol. Comm-12, 1967.

/3/ Communicating Sequential Processes. C.Hoare, Comms ACM, vol.21, No.8, August 1978.

/4/ Distributed fault tolerant computer system. D.A.Renels, IEEE Computer, March 1980.

/5/ Hardwired Resource Allocators for Reconfigurable Arcitectures. B.D.Rathi, A.R.Tripathi and G.J.Lipovski, Proc.Int'l Conference on Parallel Processing, IEEE, August 1980.

/6/ A topology reconfiguration mechanism for distributed computer system. M.Bozyigit and Y.Paker, The Computer Journal, 25, No.1, 1982.

/7/ Introduction to the Configurable, Highly Parallel Computer. L.Snyder, Computer, January 1982.

/8/ "Why Systolic Architectures?. H.T.Kung, Computer, Vol.15, No.1, January 1982.

/9/ On the design of algorithms for VLSI systolic arrays. D.I.Moldova, Proceedings of the IEEE 71, No.1, January 1983.

/10/ Flexibile Architecture Microcomputer Design. E.Zager and D.Tabak, Microprocessing and Microcomputer Design 11, 1983.

/11/ Reconfigurable architecture for VLSI processing arrays. M.Sami and R.Stefanelli, National Computer Conference, 1983.

/12/ Computer Architectures and Parallel processing. K.Hwang and F.Briggs, McGraw-Hill, New York, 1984.

/13/ OCCAM - an overview. D.May and R.Taylor, Microprocessors and microsystems, Vol.2, No.2, March 1984.

/14/ Concurrent VLSI Architectures. C.L.Seits, IEEE Trans. on Computer, Vol.c-33, No.12, December 1984.

/15/ The transputer implementation of occam. D.May and Shepherd, Proceedings of the international conference on fifth generation computer systems, ICOT, 1984.

/16/ On Supercomputing With Systolic/Wavefront Array Processors. Kung S.Y., Proc. IEEE, July 1984.

/17/ Post-failure reconfiguration of CSP programs. M.Shatz, IEEE Transaction on Software Engineering, Vol.SE-11, No.10, October 1985.

/18/ An Engineerable and Reconfigurable Cellular Array Processor. J.M.Cotton, Parallel Computing 85, 1986.

/19/ Hierarchical array processor (HAP)

featuring high realibility and high system performance. T.Ishikawa, S.Momoi, S.Shimada, Y.Ogawa, Proc. of the 1986 International Conference on Parallel Processing, August 1986.

/20/ Product Information, The Transputer Family. Inmos Corporation, Part No.72, 1986.

/21/ Synthesizing non-uniform systolic designs. C.Guerra and R.Melhem, Proc. of the International Conference on Parallel Processing, IEEE, August 1986.

/22/ Hardware reconfiguration of Transputer networks for distributed objekt-oriented programming. D.Q.M.Fay and F.K.Das, Microprocessing and Microprogramming 21, 1987.

/23/ Systolic Arrays-From Concept to Implementation. J.A.F.Fortes and B.W.Wah, Computer, Vol.20, No.7, July 1987.

/24/ Mapping Data Flow Programs on a VLSI Array of Processors. Mendelson B. and G.M.Silberman. Proc. 1987 Int'l Conf. Computer Architecture, June 1987.

/25/ Wavefront Array Processors-Concept to Implementation. S.Y.Kung, S.C.Lo, S.N.Jean and J.N.Hwang, Computer, Vol.20, No.7, July 1987.