HIGH PERFORMANCE COMMUNICATIONS IN PROCESSOR NETWORKS

Jesshope CR, Miller PR,Yantchev JT

Dept. of Electronics and Computer Science
The University
Southampton
England

Abstract

In order to provide an arbitrary and fully dynamic connectivity in a network of processors, transport mechanisms must be implemented, which provide the propagation of data from processor to processor, based on addresses contained within a packet of data. Such data transport mechanisms must satisfy a number of requirements - deadlock and livelock freedom, good hot-spot performance, high throughput and low latency. This paper proposes a solution to these problems, which allows deadlock free, adaptive, high throughput packet routing to be implemented on networks of processors. Examples are given which illustrate the technique for 2-D array and toroidal networks. An implementation of this scheme on arrays of transputers is described. The scheme also serves as a basis for a very low latency routing strategy named the *mad postman*, a detailed implementation of which is described here as well.

Keywords: Deadlock-free, packet routing, low latency, routing networks, transputers.

1. Introduction

It has become clear over the last decade, that parallelism in the form of replication has been able to provide cost effective improvements in computer performance. Replication is cost effective, because it allows relatively slow but dense technologies such as MOS to compete with intrinsically faster technologies such as ECL. The technique of replication is most successful if the replication factor is high and the unit of replication is simple. To apply the high density technologies to improving the speed of computer systems, a technique is required to design incrementally extensible architectures so one can increase the processing power of a system by simply plugging in more chips. Message passing computers, collections of processing nodes connected by a communications network, offer a solution to the problem of building extensible computers. These concurrent computers are extended by adding processing nodes and communication channels.

The design of an efficient general communication network is one of the most important issues in architectures that permit any processor to communicate with any other. One choice is a network of reconfigurable topology implemented by configuring a circuit switch between communications links connecting the processors. However, even a fully reconfigurable network has limitations imposed by its static nature, fixed valency and central control; it is also very expensive to implement with the current 2-D (even 3-D) technology for networks with a large number of processors. If the nature of the problem is such that the topology required changes dynamically, then other means must be used in order to realise a dynamic virtual topology.

In order to provide a fully dynamic arbitrary connectivity in a fixed valence network of processors, a transport mechanism must be implemented, which provides the propagation of data from processor to processor, based on addresses contained within a packet of data. In such a network each processor might wish to send a message to any other. Such a data routing mechanism must satisfy a number of requirements depending on the particular application. Among the most demanding ones are:

- the routing protocol must be deadlock free;
- no packet is infinitely delayed in the network;
- a packet always takes the shortest route to its destination;
- the routing mechanism must adapt to traffic conditions and exploit the full available communications bandwidth (no restriction on the routing must be imposed other than the above);
- a node must not be able to refuse the input of a message from its user for ever;
- the highest possible throughput must be achieved (possibly only limited by the bandwidth of the internode links);
- the lowest possible latency must be achieved.

The requirements for high throughput and low latency in communication are extremely important if the ratio of computation/communication costs are to remain well balanced even in the case of fine grain parallelism. This ratio scales rather badly in highly replicated designs because of increases in the network diameter or because of wiring costs. Deadlock freedom is a natural requirement unless one wants the routing network to have the potential of being nondeterministically brought to a complete halt until some external intervention brings it back into life. The requirement for fair allocation of routing resources ensures that no packet is indefinitely delayed in the routing network before being delivered at its destination.

Different schemes have been proposed by others [1,2,3], but none of them seems able to satisfy all the requirements above. Here we discuss some of these schemes and describe a new scheme which meets these requirements and provides for a great deal of design freedom to make compromises in order to achieve a desired cost/performance ratio. This scheme can be applied to arbitrary networks; however, we will restrict our attention to k-ary n-cubes and to 2-D toroidal arrays in particular.

A method [4] is proposed to solve the problems of deadlock and livelock freedom. It provides for a maximum distribution of the processing of packets at each node thus allowing for a maximum throughput to be achieved. It is based on splitting the physical network into a set of independent directed-cycle free *virtual networks*, each one of which can be proved deadlock free and hence the whole set must be deadlock free. The proposed method allows for all available communications bandwidth between the source and destination of a packet to be utilised and an adaptive routing strategy to be implemented. It also serves as a base for the development of a very low latency and high throughput routing strategy which we named the *mad postman* [4,5].

2. Overview of the Existing Packet Routing Algorithms

Many deadlock-free routing algorithms have been developed for *store-and-forward* computer communications networks [2,3]. In store-and-forward, each packet is stored completely in a node and then transmitted to the next node. These algorithms are based on the concept of a *structured buffer pool*. The message buffers in each node of the network are partitioned into classes, and the assignment of buffers to messages is restricted to define a partial order on buffer classes. The main disadvantage of this method of routing is the very high latency in delivering an individual packet; it is proportional to the product of the packet length and the number of channels traversed. It also imposes stringent requirements on the minimum buffer space in each node. We have simulated permutations on a two-dimensional 1024 processor array, where for one packet per processor injected into the network, a maximum buffering requirement was many tens of packets [6]. A node must therefore be able to accommodate a number of full length packets, or it will fail to use its critical resources - the connections to other nodes.

Instead of storing a packet completely in a node and then transmitting it to the next node, *wormhole* routing [7] operates by advancing the head of a packet directly from incoming to outgoing channels. Only a few control digits (flits) are buffered at each node. A *flit* is the smallest unit of information that a queue or a channel can accept or refuse.

As soon as a node examines the header flit (flits) of a message, it selects the next channel on the route and begins forwarding flits down that channel. As flits are forwarded, the message becomes spread out across the channels between the source and the destination. It is possible for the first flit of a message to arrive at its destination node before the last flit of the message has left the source. Because most flits contain no routing information, the flits in a message must not be interleaved with the flits of other messages. Thus, when the header flit of a message is blocked, all of the flits of a message stop advancing and block the progress of any other message requiring the channels they occupy.

A solution to the problem of deadlock freedom in wormhole routing has been proposed in [1]. It is based on preventing cycles in a channel dependency graph. Given an arbitrary network and a routing function, the cycles in the channel dependency graph can be removed by splitting physical channels into groups of virtual channels; each virtual channel having its own queue. The virtual channels approach restricts routing of packets; it reduces the number of possible paths that a packet may take from one which may be very large to a single path, thus disallowing any adaptation to local traffic conditions.

It is claimed that to prevent deadlock in wormhole routing one must restrict the routing of packets [1]. The next section will show that this is not true. The method of *virtual networks* proposed here avoids deadlock by preventing cycles in the directed communications graph; there is no restriction on the number of paths a packet may take, given that it always moves closer to its destination. The method of virtual networks also reduces the required minimum buffer space at each node; instead of each virtual channel having its own queue, now each virtual network only needs its own queue.

Virtual cut-through [8] is a method similar to wormhole routing. It differs from wormhole routing in that it buffers messages when they block, removing them from the network; this improves the hot-spot performance of the network and, in general, increases the throughput. The deadlock properties of cut-through routing are identical to those of store-and-forward routing (i.e. it implements the same deadlock avoidance algorithms).

The method of *double-buffering* [9] is an exotic approach to deadlock avoidance proposed recently. In this method messages start off by being buffered at their sending node. Each then attempts to establish a connection with the destination node by forming an unbroken path across intermittent nodes. If a block or a failure is encountered along the route the message gives up by recoiling back to the sender, thus avoiding deadlock. After non-deterministic delay the message tries again. Once a connection is made the entire message is transferred from source to destination, where it is again buffered before eventual off-loading to the receiving system.

In [8] it is claimed that the method provides a reasonable performance, but it is likely that it may be very expensive in terms of communications costs, which will increase disproportionately as network diameter increases. It is very difficult to imagine this method working efficiently in a network of hundreds or thousands of nodes. It is also difficult to see how, in such large networks, the delivery of a packet may be guaranteed; it seems that a packet may indefinitely try to find its way to the destination node and always fail.

3. Virtual Networks - a Method for Deadlock Avoidance in Packet Routing Networks

A method is outlined here for designing deadlock free packet routing networks, which is described in more detail in [4].

Deadlock occurs in a concurrent network when no further action can take place. This is usually because, even though each component process is in a state in which it can communicate, its potential communications are blocked by its neighbours. This is a common problem in concurrent systems and is unique to them. A very good treatment of the problems of deadlock is given in [10], with methods for deadlock analysis applicable to a large class of networks. One of the most fundamental results of the work described in that paper is given by the following theorem.

Theorem 1 [10] If V is a busy, triple disjoint and strong conflict free network, then any deadlock state contains a cycle of ungranted requests (a cycle of at least three processes, each of which is blocked by the next). #

Not all cycles of ungranted requests lead to deadlock, they are just symptomatic of deadlock; hence, one approach to deadlock avoidance is to avoid cycles of ungranted requests. In packet routing networks comprised of buffer processes, cycles of ungranted requests leading to deadlock arise when all buffers in the cycle are full and each one is trying to output a message along the cycle. These cycles of ungranted requests correspond to cycles in the directed communication graph of the network and therefore, an acyclic network of buffers must be deadlock free. A more formal definition and proof of this is given as theorem 2 in [4].

Now, if a physical network can be split into a set of independent virtual networks, all of which satisfy the conditions of theorem 2, then the whole network will be deadlock free. To illustrate this let us consider the 2-D array case.

The 2-D array. Although each node of the physical grid must route messages in all four directions, this provides for a large amount of unnecessary connectivity, as far as a particular packet is concerned. All packets can be divided into four classes according to the directions they need to be routed. These four classes correspond to the four quadrants of a 2-D plane:

Class I	(+X,+Y)
Class II	(-X,+Y)
Class III	(-X,-Y)
Class IV	(+X,-Y)

Four independent virtual networks, each routing messages in one of the four quadrants, can provide for the necessary routing paths provided each packet is initially injected into the proper network. These virtual networks represent four systolic arrays, as shown in Fig.1. Each node is a buffer process of the type described above. These systolic arrays all satisfy the conditions of the theorem presented in [4], and the whole network is therefore deadlock free by design.

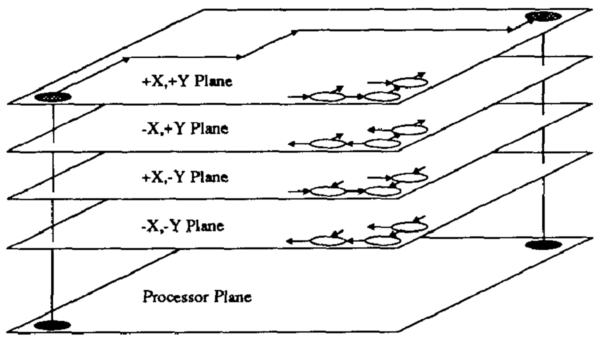

Fig.1 Virtual Network Topology; showing a message routing in the +X,+Y Plane

Toroidal networks. For a toroidal network, at a first glance, it seems impossible to close a mesh in such a way so as to allow for communications between opposite ends of the mesh, without introducing any directed cycles in the communication graph. However, by stating the problem in a convenient form, a simple solution which provides the connectivity of a torus among the physical nodes and preserves the deadlock freedom of the whole network is possible.

If the [N x N] array is split into four [N/2 x N/2] subarrays (if N is odd then round off suitably), then the arrangement in Fig.2 provides all the connectivity of a torus. In this case the classes of message have to be further subdivided, so that if the distance travelled by a packet in either direction (d say) is greater than N/2, then it is routed in the opposite direction, by a distance of d-N/2 using the wrap-around connections. In this way a total of 8 message classes or virtual networks may be defined to provide deadlock free packet routing in the 2-D torus.

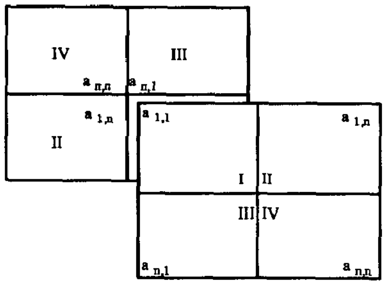

Fig.2 Unwrapping a Torus

This arrangement needs a storage space at each node for eight (8) packets. This though is not a large number and seems a reasonable price to pay for a twofold reduction in the network diameter and a peak performance at each physical node of eight packets in and eight packets out.

Mapping virtual networks onto a physical network. If buffering is not a restriction the nodes of the systolic arrays (the virtual nodes) may be suitably mapped onto the physical nodes of the physical array, so that packets from virtual planes may be interleaved over the same physical links. In this case, to preserve the deadlock freedom, flow control must be used with each output on a virtual channel being immediately followed by a wait for acknowledgement. This scheme is illustrated in Fig.3. In store-and-forward and virtual cut-through schemes, to reduce the overhead of the acknowledgement signals, it is possible to use one acknowledgement per packet rather than per data item. In the case of wormhole routing the acknowledgement must be per data item, otherwise deadlock may result.

Application to arbitrary networks. It is easy to see that this method can be applied to arbitrary networks. In the extreme case, each message class will consist of one message only and the number of virtual subnetworks will be equal to the number of classes. Although possible,

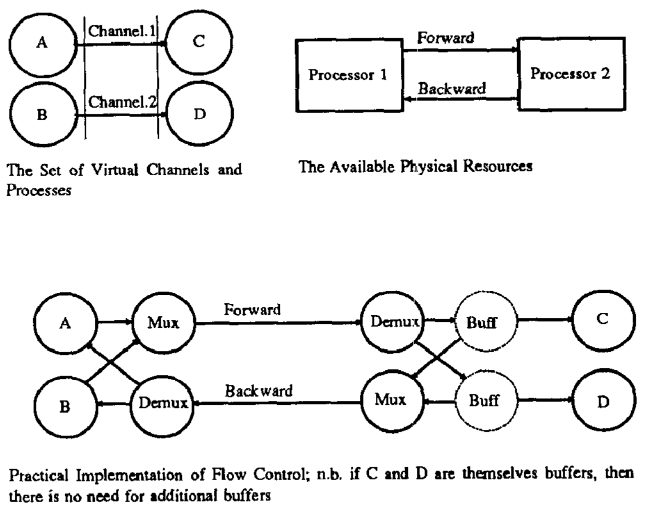
A
B
C
D
Channel.1
Channel.2
The Set of Virtual Channels and Processes
Processor 1
Processor 2
Forward
Backward
The Available Physical Resources
A
B
Mux
Demux
Forward
Backward
Demux
Mux
Buff
Buff
C
D
Practical Implementation of Flow Control; n.b. if C and D are themselves buffers, then there is no need for additional buffers

Fig.3 Deadlock Free Mapping of Two Virtual Channels over one Bidirectional Link

in such extreme cases the method may not be of great use. For example, some ill conditioned networks may require excessive storage requirements, but then one can always reduce the packet size or indeed use wormhole routing. In most practical cases, however, the number of virtual networks need not be large.

For arbitrary networks the procedure below must be followed:

i) subdivide all messages into classes according to their direction of routing;

ii) split the whole network into directed-cycle free virtual subnetworks, each of which provides all possible paths (if needed) for any of the message classes in (i);

iii) implement a buffer process at each node of the virtual subnetworks and map them, together with all virtual channels, onto the physical network.

An advantage of this method is the design freedom it offers. The designer can always make a trade-off between storage space, speed, number of possible paths, etc.. For example, a packet routing protocol for a 4-D cube can be designed in at least two different ways. Firstly, the 4-D cube can be split into sixteen 4-D systolic cubes (e.g., a 4-D cube routing chip used), thus utilizing all possible paths the physical network provides. Alternatively, the 4-D cube can be split into two cascaded 2-D subnetworks (say, two cascaded 2-D cube routing chips) each in turn split into four systolic arrays. Each packet is then routed in the first two dimensions and then in the next two. The first network strips off the first two addresses of each packet before passing them to the second subnetwork, which then treats the next two symbols as addresses in the next two dimensions and routes packets accordingly. This reduces the number of possible paths to some extent, but still allows for adaptive routing.

4. An Implementation of Deadlock-Free Packet Routing using Virtual Networks on a 2-D Array of Transputers

The transputer [11] was the first commercially available microprocessor that combined both processing and communication facilities on a single chip. However, it only provides for fixed point-to-point communications, rendering it most useful for embedded systems (its intended application). The more general packet routing mechanism could be implemented by a dedicated software layer. Running this together with applications on a single transputer would result in a poor performance and bandwidth utilisation, due to the internal sequentiality of the transputer.

For maximum performance one may provide a dedicated routing transputer plane, coupled to a processing plane of transputers. Because of its bidirectional links one transputer can implement two virtual planes without any need for channel multiplexing (Sec.3); because of the independence of the two channels within a link the virtual planes remain independent. In this way the four virtual networks for a 2-D array connectivity can be implemented on two dedicated transputer routing planes. Alternatively, the four virtual networks can be incorporated into a single routing plane by using channel multiplexing.

The transputers in the routing plane(s) are fully connected and therefore a fifth link must be added to couple them to the processing plane. This fifth link can be realised by using the memory interface of the routing transputers.

A transputer system that utilises two T414 routing planes and a single T800 processing plane is currently being implemented using surface-mount chiprack technology.

5. The Mad Postman - a Low Latency Routing Strategy

Among our top requirements detailed in the introduction were those for adaptive routing, high throughput and low latency. It is easy to see that the method proposed above provides for a fully adaptive routing strategy. Also, the processing of packets can be fully distributed over all input and output channels and thus all available channel bandwidth may be exploited.

Here a routing strategy, described in [4,5], with the same channel bandwidth, reduces the latency of routing packets by more than one order of magnitude in comparison to the other known present time routing strategies such as store-and-forward, wormhole routing and virtual cut-through. This strategy was named the *mad postman*, which will be seen to be quite an appropriate name.

Latency is defined here, as elsewhere [1], as the delay in delivering a single message in isolation. In the treatment below bit-serial internode communication will be assumed. The latency of the present known routing strategies needs to be defined in order to provide comparison with the *mad postman*. First define:

L is the packet length in words(flits);

W is the word length;

T is the time to transmit one bit;

D is the number of channels traversed;

then,

i) store-and-forward routing, latency = LWTD;

ii) wormhole routing, latency = WTD + LWT;

iii) virtual cut-through, latency = WTD + LWT.

Store-and-forward routing gives a latency that depends on the product of the message length L, and the number of communication channels traversed D. Wormhole routing and virtual cut-through result in a latency which depends on the sum of the two terms. Virtual cut-through provides for the same latency as wormhole but eliminates the deteriorating effect of blocked messages.

Obviously, for minimum latency one must route along the dimension of the leading address flit. Latency will be further reduced if when each packet has been fully routed in a particular dimension then the associated address flit is stripped off. This will also increase the efficiency of transmitting packets. If a packet cannot be routed in the first dimension then it can either be routed in some other dimension or wait till the necessary channel is available.

To summarise, if virtual cut-through routing with all features mentioned above is implemented, then the minimal achievable latency would be WTD + LWT, and accordingly the delay per node would be WT seconds. For example, in case of asynchronous communication with eight data bits, one start bit, one stop bit and T=50 nSec., it means a delay per node of at least 500 nSec. This delay is necessary to accumulate enough information (a full address flit) for a routing decision to be made.

Another important consideration is a dedicated routing hardware. If such is available, then there is nothing inherently wrong in using it for any purpose if it would otherwise stay idle. Furthermore, there is nothing wrong if some (or even all) of the additional work turns out to be useless, if this helps us achieve a latency far less than the minimum latency achievable otherwise. This is the strategy adopted in *mad postman* routing. In order to reduce latency, without changing any of the terms L,W,D,T, the only choice left is to output every packet along the same dimension it is arriving from, as soon as its first bit arrives and make the routing decisions later. Latency then becomes:

iv) mad postman routing, latency = kTD + LWT

where k is a coefficient between 0 and 1 and is implementation dependent. In the case of synchronous communication k is most likely to be 1, and in the case of asynchronous communication kT will be the time to restore the pulse waveform (usually far less than T). When the whole leading address flit has been examined, one of the following will be true:

- the leading address flit indicates that the packet has to be forwarded further along the same dimension. Then simply continue transmitting the packet, thus achieving minimum latency.
- the leading address flit indicates that the packet has been fully routed in this dimension, and now has to be routed in the next dimension. In this case one must stop the transmission along the first dimension and start transmitting the second flit along the next routing dimension. The second address flit need not be delayed for more than kT time and again minimum latency is achieved! The penalty is that a 'dead' address flit has been transmitted along the first routing dimension.
- the packet has arrived at its destination. Then stop transmission and store the rest of the packet into memory. Again a 'dead' address flit has been transmitted.

Using the figures quoted in the example above, the mad postman will result in a delay per node of less than 50 nSec. which represents an improvement of ten times achieved without any change in the available channel bandwidth. This is indeed a significant step towards reducing the communications cost in a highly parallel computer.

What then, if any, are the costs of this reduction in latency. The *mad postman* only works for the class of directed cycle-free networks described above. However, as it was shown in the previous section, any network can be split into a set of virtual directed cycle free networks which can then be multiplexed onto the physical network. The only additional requirement to obtain this low latency is that the reverse acknowledge signals introducing discipline in the processing of packets be implemented in hardware.

The *mad postman* works properly in any traffic conditions and can be made to adapt to traffic density. Of course, in conditions of increased traffic some blocking is inevitable due to the inevitable contentions for output channels. If, due to blocking at some intermediate node, a packet cannot be routed in the dimension of the leading address flit then it can either wait for the corresponding output channel to be freed or be routed along some other dimension by swapping the first address flit with any other depending on the traffic conditions (i.e. adapting to traffic density). Note that no dead address flit is generated when a packet changes its dimension of routing due to traffic conditions.

In general, in an empty network the maximum number of generated dead address flits will equal the degree of the network. However, they will never reduce the performance beyond that of virtual cut-through or wormhole routing. By comparing it with the local address a dead address flit can always be recognised as such and ignored by the intermediate nodes on its way. It will either quickly reach the boundary of the associated virtual network or will be blocked at some intermediate node. In both cases it will be immediately discarded from the network. Of course, a dead address flit may affect some other packet if there is a contention for the output channel of the associated routing dimension. However, the delay will never be greater than the duration of one address flit and this is precisely the minimum delay introduced at each node by virtual cut-through or wormhole routing. By the time the first address flit has been accumulated the blocking dead address flit will have left the node and the packet may be routed properly. Since normally a whole packet is much larger than an address flit then the probability of a 'good' packet blocking a dead address flit (which will result in discarding the dead flit from the network) is much greater than the probability of a dead address flit blocking a 'good' packet. Simulation results (see later) convince us that 'dead' flits have little or no effect on overall performance. However, although not covered here, an improvement is possible whereby the 'dead' flits can be reduced in size as they propagate thus reducing their effect still further.

As traffic density increases, the number of blocked packets will also increase and correspondingly, the number of dead address flits per packet will decrease and the dead address flits will disappear more quickly from the network. In the extreme case of a maximum traffic density no dead address flits will be generated and packets will propagate in virtual cut-through or wormhole fashion. However, due to the much lower latency of the *mad postman* and hence better relaxation properties of the routing network, it will allow for higher throughput to be achieved as well. The improved latency also increases the channel bandwidth usage by increasing the rate of processing of packets by the communication network.

In general, the *mad postman* is most advantageously used in lower dimension networks, for example 2-D meshes, where latency is usually higher due to higher network diameter. Moreover, such networks best match form to function, and in current implementation technologies (i.e. VLSI chips and PCBs) they will lead to a cheaper hardware, thus allowing for a higher cost/performance ratio to be achieved. It may appear, that in higher dimension networks the efficiency of the *mad postman* routing will be lower due to the higher number of dead address flits generated per packet. However, in such networks the dead address flits will be shorter and (for a given number of nodes N) the total number of dead address bits will be the same - $\lceil \log_2 N \rceil$ (the start and stop bits aside).

If well implemented, the *mad postman* will provide a message latency which is the absolute minimum achievable for its class (eg. synchronous or asynchronous networks) In an empty network, the speed of propagation of a packet will only be limited by the physical constraints of propagating a signal pulse (electrical, optical, etc.) between the source and the destination.

6. Hardware Implementation of the Mad Postman Routing Strategy

In this section we present an overview of a prototype hardware design that incorporates many of the ideas presented so far. The objective has been to develop a communications network, with a 4-degree mesh topology, that applies the *mad postman* routing stategy and achieves deadlock freedom through the virtualisation of acyclic routing planes.

Methods for realising the four virtual planes necessary for full interconnection from various hardware configurations have been covered, so it is suffice to say that what follows is a presentation on a hardware realisation of a single virtual plane (ie. it only routes in 2 directions).

The basic configuration. At each node in the network there are two machines, namely X and Y, each with its own datapath, finite-state machine controller and buffers. Each has its own unique data input from the previous node in the corresponding dimension, but they contend for the respective output channels depending on behaviour.

The datapath. Before discussing the datapath organisation, consider the format of packets adopted in the system (Fig.4). There is either one or two leading address flits, followed by the data to be transmitted. Economies of buffer size at each node will dictate the maximum packet length that can be in the system, but within this upper bound variable length packets can exist in the system simultaneously.

The information is segmented into units of eight-bit bytes (or flits), each with its own header bit. The address flits contain a dimension flag indicating whether the address is the first of two, or the second. In the case of only one address flit preceeding the data, the interpretation is that it is the second address. Another interpretation of the flag's meaning is that it indicates whether or not the packet is in its final dimension of routing.

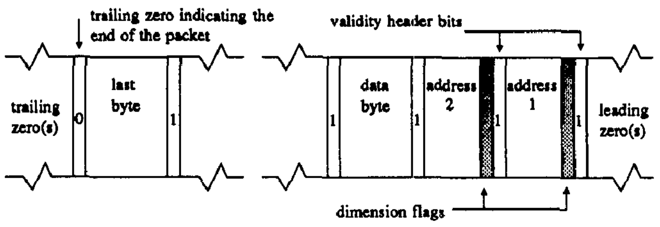

Fig.4 Format of Packets in the Network

The reader will notice therefore that only seven bits are available in the current scheme for addressing. Thus, at most a 128x128 array can be supported. Furthermore, because of the bit-serial nature of operation, the flit length can easily be increased to theoretically any size by simply *stretching* the datapaths. The machine operation will remain fundamentally unchanged. The end of a message is detected by a zero validity bit following the last databyte of the packet.

In each of the machines at each node therefore, with the current packet format, a 9-bit datapath shift register must be provided so that flits can be held in their entirety for decision and buffering purposes. The essence of the *mad postman* routing strategy is introduced here. Consider the schematic comparison (Fig.5) of an obvious datapath insertion and the one adopted. The communication delay incurred by a message in traversing nodes is dramatically reduced (k=1). Therefore, latency is also minimised, being paid for by eagerly transmitting the data (ie. before any decision can be made concerning its address). By the time a leading address flit is fully into a datapath at say node X, then it will be partially into subsequent datapaths up to node X+8 (assuming no blockage). Hence the generation of dead address flits as discussed earlier.

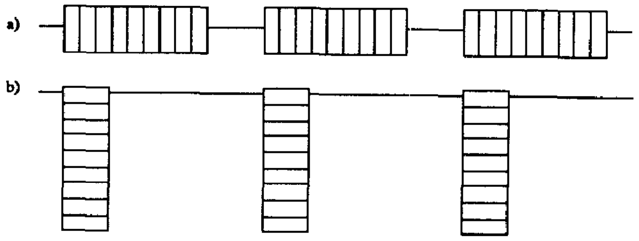

Fig.5 Comparison between a) conventional datapath (wormhole) & b) mad postman datapath

At a more detailed level of implementation, Fig.6 shows the datapath arrangement. The source and buffer interface registers connect directly to the 8-bit transit buffer for the machine. Having a separate source register, as opposed to directly loading into the datapath register, enables a machine to accept and transmit separate packets simultaneously. Indeed, if sourcing can occur sometime before a message has finished buffering then we are *chaining* the packet through the buffer ensuring maximum throughput at all times. Each time a valid flit sits within the datapath a copy is made to the buffer interface register ready for buffering if required.

Switching occurs at the output rather than the input. The disadvantage of the latter approach being that, in the case of dimension transfer into the complementary datapath (through junctioning or adaption), both input channels are occupied.

Adjacent node interface. Between adjacent machines there are two signals; data transmission forwards and a busy signal in the reverse direction. Because of the exclusivity of the these signals *(if the next node is busy then data o/p from the previous node is not required as it will either have to be re-routed into the other dimension or buffered)*

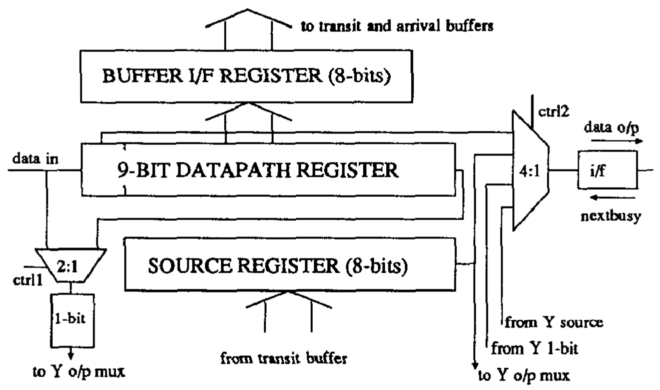

Fig.6 Datapath Organisation

they can share the same physical link as shown in fig.7. The important feature to note here is the symmetry of the circuit. With very simple reconfiguration control the sense of the two signals can be reversed. The implications are of course that, with negligable overheads, the same hardware network can be used to emulate the four virtual networks. Alternatively, two identical hardware networks could be implemented, thus sharing the virtual load, or even one network for each virtual plane (or more!). The trade-off is near-linear and need not stop here. We argue that the independence of virtual planes is a more flexible way of obtaining bandwidth than adopting bit-parallelism.

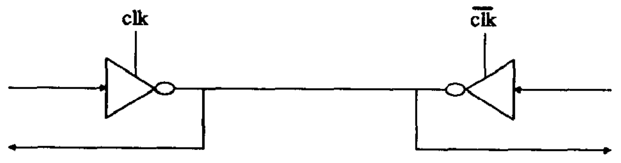

Fig.7 The Internode Link

The finite-state machine controllers. The controlling machines across the array are not functioning in lockstep as would be the case if the datapath insertions were 9-bits. They have been implemented as autonomous FSM's, activated by the head of an incident packet and deactivated by the tail. The states of the machine, one for X and one for Y, at each node are as follows:

Sourcexx: Output X buffer data onto X output channel;

Sourcexy: Output X buffer data onto Y output channel;

Junctionx: Re-route the 2nd address byte and subsequent data;

Adaptx: Swap the address bytes, incurring a delay of 9 cycles, and re-route into Y;

BFAX: Buffer an arrival in X arrival buffer;

BTHX: Buffer a blocked message in X transit buffer;

BTOX: Buffer a message, that wishes to junction into Y but can't, in X transit buffer.

Fig.8 shows the state diagram (complementary states exist for the Y machine).

In the case of a junction being established, subsequent input data is directed to the complementary output, with the first address flit escaping along the same dimension. Adaption of routing (for a packet with two addresses) occurs if a path is blocked and the complementary output is free. If it is not, or the message is in its final dimension then it must be buffered (BTH). The BTO state is essentially the same as BTH. It differs only in that the buffered data must be output into the complementary dimension upon re-sourcing. Adaptive sourcing can also

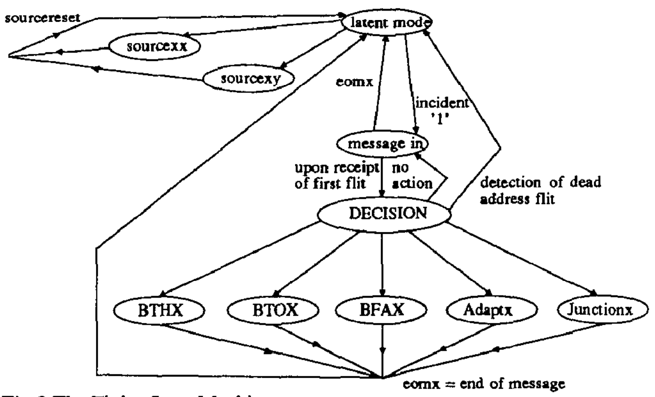

Fig.8 The Finite-State Machine

result in BTH buffered data being output into the other dimension. The conditions for this to occur are that two addresses have been buffered, and only the complementary output is free for sourcing. Otherwise, the buffered packet will be re-sourced into the same dimension it entered the node on.

The reader will note that there are two adaption mechanisms, namely adaptive routing and adaptive sourcing. They are not essential to the correct delivery of packets, but are simply system performance enhancements. This is verified through our simulation results, although one can intuitively realise their benefits. It should also be noted that the hardware expense of their inclusion is negligable.

Buffer Protocol. In the initial development of a suitable buffering protocol for the system, the approach taken was to provide, intuitively, the least performance-limiting solution. The objective being to provide a reference upon which to compare simpler implementations through simulation. This we termed the *ample buffering* strategy.

When a machines buffers are occupied it must generate a *busyback* to the previous node in its dimension to prevent reception of packets that it could potentially not cope with. In the ample buffer case therefore, no *busybacks* are ever generated. In the more realistic case of each machine providing only a single buffer then *busybacks* are generated alot more frequently.

A point worthy of note with minimal buffering systems is that when a region in the array is heavily loaded, *a hot-spot,* the generation of busy signals due to buffer occupation will be more prevelant and will in effect create a protective barrier. The result being that incident 2-dimensional packets will if possible adapt around the area, thus preventing worsening of the situation. This is viewed as an attractive feature that would not occur with ample buffers and occur less often with multiple buffers.

The provision of multiple buffers at a machine, as well as being expensive in chip area, also involves greater complexity in the buffer control hardware. For these reasons, and those stated above, our hardware interests are currently focussed on a minimal one buffer per machine implementation that enables variable length packets to co-exist in the network. This shall now be described briefly. Note that each machine has an arrival buffer, with its own input pointer, that is independent of the transit buffer and whose operation is quite trivial.

The transit buffer has two pointers; one for input (buffering) and the other for output (sourcing). Upon buffering a blocked packet, a flag is set to indicate that the buffer will contain a packet, and the flits are written to the buffer as they arrive, incrementing the pointer each time. When the conditions are satisfied for sourcing, be it adaptive or not, the buffered packet is output in the order in which it arrived (in the case of adaptive sourcing this rule is broken for the two address flits), and the flag reset. This can occur at any time after the first flit is buffered, since the input and output bandwidths are identical, and the output pointer will therefore always remain behind the input pointer for that packet. Furthermore, as soon as sourcing commences, the node can accept another packet (by disasserting its *busyback*, which is the flag set and reset as above) and if necessary buffer this behind the source pointer. In other words, maximum throughput is ensured by allowing the *chaining* of blocked packets through the transit buffer, without the possibility of loosing information.

To enable the co-existance of variable length packets in the network, information needs to be derived that indicates the length of each individual packet buffered. This is readily available from the input pointer value at the end of the buffering process, so all that is required is to latch this value before the pointer needs to be reused, and sourcing of the packet has exceeded its length. This latched value is then compared with the output pointer to indicate the end of sourcing.

Because of the *chaining* ability of the buffer it is in fact necessary to provide two latches which are used alternately for comparison with the output pointer value. Fig.9 shows a schematic of this simple but effective arrangement. The two latches MSLEN1 and MSLEN2 are initialised to maximum (all 1's) and, when their use is over, reset to maximum. They alternately latch the value of PTADDRX at the end of buffering. The source pointer is initially compared to MSLEN1, and then the comparison alternates upon re-sourcing of subsequently buffered packets. The comparison is used to terminate the sourcing and toggle the comparison selection, while the derivation of the latch and reset signals is trivial. Of course, there are issues not discussed here, such as resolution of simultaneous read/write's to the single-port RAM buffer, and the orchestration of reading and writing, but hopefully the

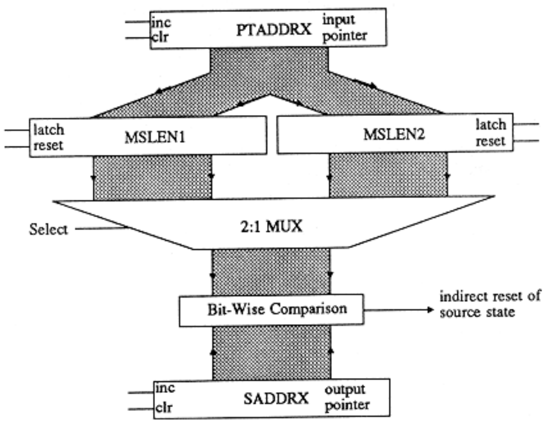

Fig.9 Buffer Pointer Arrangement allowing variable length packets

Processor - communication node interfacing. Here we shall only consider the one-buffer machine, as this is the probable implementation of the family.

There exists a time in the machine's operation, which can be decoded from it's states, when the buffer contents can simply be overwritten by external means. This is how information may be injected into the network, and it can occur at any suitable time during the networks activity. The input pointer can be used by this injection mechanism to write data to the transit buffer. However, this scheme does not decouple the communication network from the processing element very well. A better implementation would employ a separate buffer for the injection/reception of packets, as depicted in Fig.10. Depending on the construction of the node and the injection rates required, it is possible to use many known techniques for interfacing to the communications node. These include DMA or cache-like interfaces, which could even provide hardware addressing across the network.

Hardware simulation. A full logic simulation of a 32x32 array of communication nodes, employing the described buffer strategy has been performed. It runs on the ideally suited AMT Distributed Array Processor with a Sun 3/160 host. The host software, through extensive use of the windows facilities, allows interactive buffer editing and viewing to enable the initialisation and monitoring of different routing

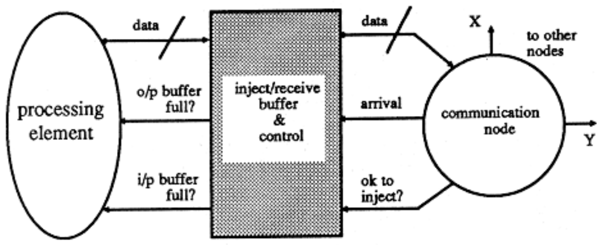

Fig.10 Processor/Communication Node Interface

scenarios. Furthermore it has the ability to display on the DAP monitor the states of the network nodes in real time. The progression of messages through the network can therefore be viewed. Clock cycle time in the simulation program is less than 0.2 seconds, although this is far from being fully optimised.

Performance. Given current technology and bit-serial communications then it is difficult to perceive a more efficient alternative scheme to the *mad postman* implementation. Simulation results are currently being obtained, and comparisons made with other known strategies. The results will be published in full at a later date [12]. However, given below are some figures which may be of interest.

To make comparisons of the mad postman with another strategy, we have also simulated virtual cut-through routing. In this, the forwarding of a packet is delayed until a whole address flit is received, and only then is a routing decision made. The comparison between the two strategies is therefore fair, since the machine behaviour is otherwise the same (adaption is still possible), and the packet format is identical.

Consider the transmission of a packet on a 32x32 node array from node (1,1) to node (32,32) of an 8 bit integer:

Mad Postman : Time taken = 89 clock cycles

Virtual Cut-Through : Time taken = 570 clock cycles

In both cases the packet is of length 27 bits.

As an example that illustrates the performance of the *mad postman* under heavy traffic conditions, we present results for the transposition of a 32x32 8-bit integer matrix, using two hardware planes to emulate the four virtual acyclic planes. The sparsity value refers to the percentage of the permutation actually performed. So, in the case of 25% sparsity, only 1 in 4 of the matrix elements are actually routed. Note also that these figures are for a non-toroidal array:

Sparsity :	100%	50%	33%	25%
Mad Postman :	464	272	191	154
Virtual Cut-Through :	664	570	570	570

It should be noted that the results given for either strategy do not include the minimal overhead of writing the packets into the buffers initially, or the corresponding overhead of reading the arrivals.

We wish to make two comments on the above results. As the traffic density is increased the mad postman latency assymptotically increases towards that of virtual cut-through. In the limiting case of maximum traffic density at all times, the two strategies will be equal. The second comment is that due to its much higher latency, even in medium to light traffic densities, virtual cut-through, unlike the mad postman, cannot utilise the available channel bandwidth and thus yield a proportional reduction in permutation time.

Future work. Currently in progress are simulations of the various arrangements discussed. Furthermore, a full-custom CMOS implementation at a scale of one node per die is underway, the objective being to assess and minimise the silicon area required. It is envisaged however that with current VLSI densities a powerful processor and four hardware networks (one for each virtual plane) could easily be incorporated onto a single chip (ie. a packet routing transputer-like device).

The current version gives k=1 in the latency equation. Before discussing a k=0 version, there exists a method, in theory, of reducing k in discrete steps towards zero. The method involves the application of phased clocks across the array, the result being that k is reduced by the factor 1/(no. of phases). In other words, in one clock cycle a packet can propagate across n nodes, where n is the number of clock phases applied. Clearly, this is a promising area for further optimisation and research is currently focussed here. Another implementation, whereby a packet is received along all or part of a dimension (k=0) in one clock cycle, is both a viable and optimal implementation of the *mad postman.* Optical technology has a place here.

Another promising field of further research is the implementation of the *mad postman* in self-timed logic.

7.Conclusions

A network partitioning has been described for networks of processors, which allows the deadlock free routing of addressed packets of data, provided that the routing algorithm used provides a minimum distance path from a packet's source to its destination. This technique has been proven for a general class of networks and has been demonstrated for a regular 2-dimensional toroidal array.

The scheme has application to concurrent hardware implementation, as it partitions the network into independent virtual networks, in which packets once injected into a virtual network will remain in it until they have been routed to their destination.

The minimum buffering requirement is only one packet of data for each node in each virtual network. In some networks, it is possible to decrease this buffering still further by reducing the number of virtual networks required to partition the network. This may be achieved by allowing packets of data to be passed from one virtual network to another, provided that no cycles are introduced by this relaxation. In such cases, a message will have to be routed to its destination in each virtual network before being transferred to another.

This scheme serves as a basis for a low latency routing strategy named the *mad postman.* The *mad postman* outputs every packet along the same dimension it is arriving from, as soon as its first bit arrives and makes the routing decisions later. A number of 'dead' address flits equal to the network dimension (eg. two in the case of a 2-D array) may be generated in the process but they will quickly disappear from the network.

If well implemented, the *mad postman* will provide a message latency which is the absolute minimum achievable for its class (eg. synchronous or asynchronous networks). In an empty network, the speed of propagation of a packet will only be limited by the physical constraints of propagating a signal pulse (electrical, optical, etc.) between the source and the destination.

Obviously, as network size increases, so do the communication costs. A common solution has been to increase the network dimensionality, thus reducing the logical network diameter and providing more bandwidth per node. This however disproportionately increases the wiring costs, physical size and physical diameter of the system. With the same channel bandwidth per node the *mad postman* achieves the same throughput, and even lower latency.

We would like to acknowledge the SERC for the support of Mr. Miller and the Bulgarian Ministry of Education for the support of Mr. Yantchev, both by Research Studentships.

8.References

[1] Dally WJ and Seitz C 1987 *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,* IEEE Trans Comp C-36 pp 547-553.

[2] Gelernter D 1981 *A DAG-Based Algorithm for Prevention of Store-and Forward Deadlock in Packet Networks,* IEEE Trans Comp C-30 pp 709-715.

[3] Roscoe AW 1987 *Routing Messages Through Networks: An Exercise in Deadlock Avoidance,* Oxford University Computing Laboratory Report.

[4] Yantchev JT, Jesshope CR 1988 *Adaptive, Low Latency, Deadlock-Free Packet Routing for Networks of Processors,* to be published IEE Proc.E.

[5] Jesshope CR, Miller PR, Yantchev JT 1988 *Programming with Active Data,* Proc. PARCELLA 88, Akademie-Verlag Berlin.

[6] Jesshope CR O'Gorman R, Stewart JM 1988 *A Microprocessor Array,* submitted to IEE Proc E.

[7] Seitz CL 1985 *The Cosmic Cube,* Comm ACM 28(1) pp 22-23.

[8] Kermani, Parviz, Kleinrock L 1979 *Virtual Cut-Through: A New Computer Communication Switching Technique.* Computer Networks 3 pp 267-286.

[9] Whobrey D et al 1988 *A Communications Chip for Multiprocessors,* Proc. CONPAR 88, Cambridge University Press.

[10] Roscoe AW, Dathi N 1986 *The Pursuit of Deadlock Freedom,* Oxford University Computing Laboratory Technical Monograph PRG-57.

[11] INMOS Ltd 1985 *The Occam Programming Manual,* Prentice Hall International.

[12] Miller PR, Yantchev JT, Jesshope CR 1989 *High Performance Packet Routing Based on Systolic Arrays,* Proc. Int. Conf. on Systolic Arrays 1989, Killarney, Ireland.