

# The MP1 Network Chip and its Application to Parallel Computers

## Short Title - The MP1 network

**C. R. Jesshope (corresponding author) and C. Izu \***

Dept. of Electronic and Electrical Eng.  
The University of Surrey  
Guildford, Surrey  
GU2 5XH  
UK  
0483 509142 (Fax 34139)  
C.Jesshope@ee.Surrey.AC.UK

\* Dept. of Computer Architecture and Technology  
University of the Basque Country  
649 p.k E-20080 Donostia  
Spain

## Abstract

This paper presents results concerning the design and testing of a fast network chip (the MP1) for parallel computers. We briefly introduce the theoretical results on which the MP1 chip design was based and describe its architecture. The chip has been fabricated and tested in small prototype system. Based on parameters measured using this prototype and a simulator implemented at the logic level we have been able to accurately model the performance of larger networks based on this chip under a variety of synthetic loads. Extensive results are presented based on these simulations.

# 1. Introduction

It is now recognised that high performance computers, i.e. those expected to reach performance rates of a Teraflop/s or more, must exploit massive parallelism. This trend is underlined by recent developments in the high performance computing market with machines such as the Intel Paragon[1], the Thinking Machines CM5[2], the Meiko CS2[3] and the recent announcement of the Cray MPP, from a company which has a traditional base in vector super computers. Typically these machines comprise up to 1K very powerful processors connected by a high performance network which provides message passing between the processor nodes. To obtain good performance which is not communication limited and which is scalable imposes quite severe constraints on network performance and three key parameters may be identified, these are network interface bandwidth, network cross-section bandwidth and message-delivery latency. The latter two have a significant impact on scalability, for example the cross-section bandwidth must scale with the number of processors in the system to efficiently solve the most general problems.

It is also widely accepted that in order to de-mystify parallel programming a programming model must be adopted which uses some form of shared memory. All that is required to achieve this is the extension of the processor's addressing mechanism into the network domain with the provision of an intelligent network interface with direct access to local memory and trapping mechanisms which convert non-local memory references into network messages. Clearly such a memory system can not be considered as a uniform random access memory as there will be differences (rather large with current network architectures) in the latency of local and non-local memory accesses, as the latter have the round trip network latency added to the basic memory cycle latency. Thus the detailed implementation may well be more problematic and will of course depend on the programming model and memory architecture, but in any model there is a paramount requirement to minimise the latency of access to remote nodes and then to tolerate whatever latency remains in the choice of architectural model. An early but still fresh example of these engineering principles can be found in the Delencor HEP machine[4]

One of the most common approaches to ameliorate the problems with high latency non-local memory accesses is the use of local caching with a message passing mechanisms to maintain cache-coherency. Such techniques are used in the Kendal Square KSR1[1], the Stanford DASH[5] and the MIT Alewife multiprocessor[6]. In such cases the unit of memory passed through the network is a cache line, which because of the overheads of cache-coherency schemes (one bit per processor per cache line in the sharing-set directory scheme) is relatively large.

Our own approach to this problem is more fine-grained[7] and requires single word messages to be passed between processor nodes. This approach is cacheless and relies on coarse grain bulk-synchronisations with data-flow scheduling to manage the relatively slow and non-deterministic network memory references. Although this combination of bulk-synchronisation and data-flow is latency tolerant, a low latency message delivery system is critical to efficient implementation. And it is for this reason that we have expended some effort in designing and implementing an ultra low-latency network architecture which will be described in this paper.

This paper presents the design space and justifies the decisions made in the implementation of the MPI network chip, the technical details of that chip and extensive simulation results from an exact simulation of a large network system based on arrays of MPI chips under various synthetic loads.

## 2. Network Design Space

There are many design decisions which must be made in choosing a network architecture, these include choice of network structure i.e. direct or indirect network, topology, message routing and flow control strategies. In addition the network must be designed so that it is free of both livelock and deadlock.

A network can be defined as a graph  $G$ , where  $G$  is a quadruple:

$$G = (N, E, I, O)$$

and where  $N$  is a set of nodes:

$$N = \{ n_i, i=1, p \}$$

and  $E$  is a set of arcs taken from the set  $EF$ , where:

$$EF = \{ e_{i,j} \ i=1, p \ j=1, p \ i \neq j \}$$

and where arc  $e_{i,j}$  connects node  $i$  to node  $j$ ; and  $I$  and  $O$  are sets of arcs which provide input and output to nodes.

$$I = \{ i_j \ j=1, r \}$$

$$O = \{ o_i \ i=1, s \}$$

the arcs  $I$  and  $O$  provide connections to the network interface at a processor node.

In a direct network there exists an input and output arc at each node in the network, such that:  $r=s=p$  and in an indirect network the inputs and output arcs are connected only to a subset of the nodes belonging to the network, such that:  $r < p$  and  $s < p$ .

Direct networks are perhaps more commonly found in massively parallel processors, although not exclusively so[2,3]. The advantages of a direct over an indirect network are the ability to exploit locality of communication, the ease of expansion and, although this depends on many other factors, the potential for lower cost implementations.

The topology of the network defines which set of arcs,  $E$ , is taken from  $EF$  and issues here are *degree* - the number of arcs incident at a node, which has a direct bearing on the cost of a network and an indirect one on performance, *diameter* - the minimum number of arcs traversed between the most distant nodes in the network and *symmetry* - the property of a network to appear the same when viewed from any node. The latter is a major issue and in many low-diameter networks based on hierarchical construction the lack of symmetry result in a non-uniform cross-section bandwidth, with some partitions of the network having little bandwidth between them. A tree is a good example of this and it is only by increasing the bandwidth as one travels further towards the root of the tree that such limitations can be tolerated. The CM-5 uses such a *fat tree*[2] and this is perhaps the most expandable indirect network. However the cost of such expansion can be considerably greater than simple scaling would suggest as wire lengths and hence delay increase with each expansion.

A growing number of networks now use low-dimensional meshes or torroids in order to minimise the cost of the node and to minimise the wire delay[1,4,5,9,10]. The disadvantages of low-dimensional networks can be attributed to their increased diameter which increases the latency and decreases the available bandwidth for non-local messages. The former arises because message delay is dependent on the number of arcs traversed, which will be related in some way with the diameter, unless the problem has a high degree of locality with respect to the topology. The latter because the cross-section bandwidth does not scale in proportion with the number of nodes in the network, again with the same locality proviso.

The take-up of low-dimensional networks has been strongly aided by advances in flow-control mechanisms in networks. Traditionally flow-control has been by store and forward, where the message is buffered entirely at a node before a routing decision is made and the message advanced further, the delay a message experiences is thus given by:

$$t = dl\tau/w$$

where  $d$  is the number of nodes traversed by the message,  $l$  is the length of the message,  $w$  is the channel width and  $\tau$  is the channel cycle time. This latency can be reduced significantly using virtual cut-through[11] or wormhole[12] flow-control in which a routing decision is made as soon as the address header has been received. Thus the delay now comprises the sum of a propagation and a bandwidth term:

$$t = (df + 1-f)\tau/w$$

where  $f$  is the length of the component of the message which contains the address, sometimes called a flow-control unit or *flit*. It was thought that this method of flow-control was optimal as a routing decision can not be made until the destination is known. However, the flow control method developed in our work[13] does even better than this as it uses an eager flow-control scheme in which the message is forwarded speculatively, as soon as the receiving node is aware of the message. The direction of propagation is taken to be the same direction a message currently travels, which for any  $k$ -ary  $n$ -cube excepting a hypercube is a reasonable strategy. This flow control scheme is called *mad-postman* and has a delay given by:

$$t = (kd + 1-k)\tau/w$$

where  $k$  is an implementation constant which is 1 if synchronous propagation is used but which can be very much less than 1 if asynchronous propagation is used. Clearly this sequence of developments has reduced the latency of message delivery by minimising the impact of the diameter by a factor of  $f/l$  in wormhole and  $k/l$  in *mad-postman* flow-control schemes. With *mad-postman* the propagation term involving  $d$  will not dominate message delay unless  $1 \ll d$

This leaves only the problems of bandwidth scaling to be considered for low-dimension networks. In any  $k$ -ary  $n$ -cube the cross-section bandwidth is given by:

$$x = (w/\tau)p^{(n-1)/n}$$

which clearly does not scale with the number of nodes  $p$ . Only for a binary hypercube is this figure within a constant factor of  $p$ , namely.

$$x = wp/2\tau$$

However, it must be remembered that the cross section bandwidth scaling in a hypercube requires a node degree which increases as  $\log_2 p$ , whereas for lower dimensional networks the node degree is constant (e.g. 4 for a 2-D mesh or torus). One strategy which can be used in order to obtain a scalable cross-section bandwidth in low-dimensional networks is to scale the degree of each node by  $p^{1/n}$  in some manner and to use these additional links to replicate the original topology.

The final issue in the design of networks is that of routing strategy, i.e. the determination of the set (or bag) of nodes that comprise the trace of a message's path through the network. Various strategies have been adopted, but when choosing a strategy consideration must be given to deadlock freedom. The simplest strategy is oblivious routing, where a message takes a predetermined path, for example by routing each dimension of a  $k$ -ary  $n$ -cube to completion in some predefined order. It can be shown that such a strategy is free of deadlock[14]. Alternative strategies allow messages to adapt to traffic conditions so that if a message is blocked when following an oblivious strategy (usually used as a basis for adaptive strategies) then the message may make use of other free arcs at the node at which it is blocked[13,15]. There is a distinction between allowing messages to adapt over only shortest paths between source or destination or whether arcs which take a message further from its destination are also allowed. The latter also introduces livelock as a design issue, for it is now possible for messages to form dynamically repeating blocking patterns.

## 3. The MP1 Message Router Chip

The main goal of the MP1 design was to build a high-performance router chip in a modest technology which would take advantage of the mad-postman flow control [13] and adaptive routing over shortest paths, since both features when taken together promise a significant reduction in latency. The implementation uses degree 4 nodes and can be configured into grid or torus networks.

### 3.1 The MP1 network node architecture

The chip internal organisation is described in detail in [16], but a brief summary is given here for completeness. Figure 1 shows the basic chip architecture. Four acyclic routing planes per chip, (+X,+Y), (+X,-Y), (-X,+Y) and (-X,-Y), provide for the deadlock-free transmission of messages. Each plane has four 3-bit wide channels, one input and one output, per dimension. When a number of MP1 chips are interconnected to form a mesh or torus, a set of four independent virtual networks is built. Each of the planes are identical in structure except for the direction of the channels.

The main components of a plane are the routing machines (RM) that forward messages incident on either of the input channels. The routing machines are data-driven and are controlled by the message header which provide both address and control information. In all eight routing machines run in parallel allowing all channels to be active simultaneously. Each routing machine comprises a controller, a 32 byte buffer, and multiplexing and arbitration logic. Figure 2 shows the plane organisation. The interface to the four routing planes comprises an injection and a reception controller providing access to the processor node via two further 3-bit wide data channels. Some form of interface chip is therefore required to perform 3-bit parallel to word parallel conversion. Up to eight MP1 chips may be attached to the same interface channel as chip and plane addressing and arbitration is supported. This allows us to scale the cross section bandwidth. The de-coupling of router and interface function allows model dependent interface chips to be developed. For example a simple message passing interface has already been fabricated and a data-flow interface is now being designed. Two further 3-bit channels provide a network bridging interface which allows multiple MP1 networks (maximum size of 1024 nodes) to be interfaced without processor intervention, these are called the piggyback channels. A data byte (8 data bits plus 1 synchronisation bit) is transferred across any MP1 channel in three MP1 clock cycles.

### 3.2 MP1 chip features

**Message format.** A message comprises a header containing routing information, followed by a number of data bytes. The header of a message comprises one or two bytes containing address and control information. The first three bits of each byte header determines the consumption mode of the message, leaving five bits to indicate the destination address. Messages are divided in one or more segments of size up to 32 bytes allowing arbitrarily sized messages. Special codes are used to indicate "end of segment" or the "end of message" conditions. Message segments are transmitted synchronously, but each segment may be transmitted asynchronously relatively to its neighbours. The first segments reserves routing machines on its route, which are not released until an EOM condition is established. A multi-segment message has the structure shown in Figure 3.

**Buffering space.** Each input channel has assigned to it a buffer of 32 bytes, so once the router starts the transmission of a segment completion can be guaranteed. A buffer containing a blocked segment will not accept any more inputs until the buffered packet resumes transmission. Blocked multi-segment messages will therefore be buffered over a number of nodes in the network and will consume routing resources whether advancing or not. This is not the case for messages of less than 32 bytes.

**Mad Postman packet switching.** The mad postman routing machine immediately forwards a message on the first three bits of the first of (possibly) two address flits. The first flit always

encodes the address of the dimension in which the packet is travelling. The machine will, if possible, forward the message in this dimension before it has the complete address. If this speculative routing decision was bad then the message has completed its travel in this dimension, the information propagated is no longer required and only the remainder of the segment is switched to another channel (determined by the control in the address header). The *dead address flits* which propagate beyond their contained address can only be stopped when they are blocked by a node whose buffer is full or being filled. Using this control-flow policy, the delay per node is just one clock cycle.

**Message routing.** The MP1 chip uses adaptive routing at the message level reducing message latency in congested networks. In the absence of blocking, a message will continue to travel in the direction it was injected (it is the responsibility of the interface to ensure that the first address flit corresponds to this direction). If the output channel is blocked however, after the first address flit has been accumulated at that node, the second flit can be turned into the alternate dimension (assuming this is free of course), followed by the first (buffered) flit, followed by the data. It can be seen that this process swaps the order of address flits so that they remain positionally encoded according to direction travelled. If both channels are blocked the first segment of the message is buffered at that node and any further segments will be blocked at previous nodes in the message's trace. If buffering occurs the packet will be re-transmitted immediately either channel becomes free.

**Network transport and consumption modes.** The routing machines are data-driven and are controlled by flits which provide both address and control information for each dimension in the network. Each address flit comprises an address in the range 0..31 and a control code in the range 0..7. The control codes determine the action of the routing machine on receipt of a message. Three of the eight codes indicate that the current message is travelling in two dimensions. These indicate point-to-point mode, advancing with static routing, advancing with adaptive routing;, or being a broadcast message. The remaining codes signify that the message has only one dimension to traverse and are used to determine the mode of consumption of the message at that routing machine. The router supports the following modes of message consumption:

- Point-to-point mode; the message data is delivered to the node interface channel.
- Broadcast mode, the message data is delivered to all node interface channels within the bounding rectangle determined by the source of the broadcast message and its destination addresses. A simple spanning tree algorithm is used which delivers the message to X, Y and interface channels if a further address follows, or to the X or Y and interface channels if no other address follows.
- Intra-plane piggy-back mode, the message is consumed by the output of the same routing machine to which it was addressed. Thus, a message that arrives on the X input channel and satisfies the destination address is consumed by the X output channel of that routing machine
- Inter-plane piggy-back mode, the message is consumed by the piggy-back channel which links planes within the MP1 chip and which may also act as a gateway between networks at the chip level.

The piggy-back modes allow further address/control flits to be nested within the data of a message which will subsequently be interpreted by subsequent nodes as a header, without processor intervention. Intra-plane piggy-back mode may be used to perform operations such as remote broadcast by embedding broadcast routing flits as the first two flits of a point-to-point message. The two sets of addresses then define the bounding rectangle for broadcast.

**Torus Networks.** The MP1 chip was originally designed to implement scalable bidimensional meshes but it can also be used to build toroidal networks. The first problem that appears when adding the wraparound links in each routing plane is that the cycles resulting in this topology will invalidate the proof of deadlock freedom. Harden and Linder[15] suggested a solution to the problem of deadlock in toruses that involves splitting each cycle using additional virtual networks or levels as shown in figure 4. In that work it was shown that the minimum number of levels required to avoid deadlock in a k-ary n-cube is  $n+1$ , three in bidimensional networks. On the other hand, in [16,17], it is shown that this does not necessarily require three times the number of routing resources required for a mesh, as not all nodes will be used in each level of the network. Figure 5 illustrates one implementation of such a wraparound scheme, following the method shown in[17]. Level 2 is devoted to messages that use two wraparound links, level 1 for those that require only one wraparound link (X or Y) and level 0 for messages that will not use any wraparound links. It can be seen that the three levels can be implemented with the resources of two, as only the nodes in the south-east quadrant (shaded in the figure 5) will ever inject messages into level 2, whereas no message coming from levels 2 or 1 into the level 0 will ever reach the south-east quadrant (also, shaded in the figure 5).

To support this toroidal implementation, some form of encoding must be used to indicate the level to which a message is to be delivered. The currently implemented MP1 chip was not designed with such a mechanism in mind but nevertheless toroids can be implemented by encoding the level using the most significant bit of each address. Figure 6 illustrates this. The 00 code indicates level 2 (LL in the diagram), 01 or 10 indicates level 1 (HL and LH in the diagram) and 11 indicates level 0 (HH in the diagram). Therefore, we must use four MP1 chips to implement this.

One problem with this scheme, which has been investigated in our simulations of the MP1 network architecture, is the injection strategy into these three levels; there will be an uneven distribution of messages injection under random traffic conditions. Any message injected at level 1 or 2 will end up being consumed at level 0, which will become the heaviest loaded plane. Apart from a fixed strategy of injecting each message in its corresponding level according to the number of wraparound links traversed, a message can also be injected in any level above this, provided it has the correct address to identify the destination node in the corresponding level. Four classes of messages can be identified according to the number of wraparound links that must be traversed:

- 2 wraparounds: must be injected in level 2, plane LL.
- 1 wraparound, X dimension: must be injected into level 1 or above, planes LL or HL.
- 1 wraparound, Y dimension: must be injected into level 1 or above, planes LL or LH.
- 0 wraparounds: may be injected into any level.

The injection strategy should be a function of the traffic pattern in order to evenly distribute the message load. However this requires global information on the status of the network. As this is not feasible a global method that presumes the network status has to be used. We have tried different algorithms for random traffic and tuned them to compensate the difference in channel utilisation. The applied scheme, described later, determines in which chip each message has to be injected.

### 3.3 MP1 chip implementation

The MP1 chip prototype has been successfully fabricated in the ES2 1.2 micron CMOS process using a cell based approach to design. It is not therefore an optimal implementation nor does it use state-of-the-art technology and yet the performance obtained is very respectable as is underlined by the simulations that follow. Full technical details can be found in the MP1 chip reference manual[18]. The MP1 chip has a die size of a little less than  $100 \text{ mm}^2$  and is housed in a 120 pin ceramic PGA package of which 106 pins are used. It uses a 5V power supply and when clocked at 25MHz, the power consumption is less than 0.7W.

Figure 7 shows an active back-plane that has been constructed using 16 MP1 chips. It allows networks of up to 16 processor cards to be connected in a single rack. The 20 pin connectors provide power, ground and interface channels to the processor cards, which must contain an interface chip of some description. Processor cards based on the T800 transputer and the Acorn ARM have been tested in this back plane.

## 4. The MP1 Network Architecture Simulations

### 4.1 Basic Simulation Guidelines

In order to evaluate networks based on MP1 routers, the chip behaviour has been emulated by a time-driven simulator running on a DAP 510 array system. A software module, written in Fortran+, emulates the MP1 routing machine architecture at the logic gate level. In each cycle the load phase updates the values of the clocked components and a second phase re-evaluates the signals derived from the combinational circuits of the chip. A cell-state monitor displays state information from all routing machines and allows us to control and check the proper functionality of the simulation process.

The DAP enables us to simulate 1024 modules in lockstep mode. It maps each RM module in a processor and supports the network interconnection by simple instructions of planar shifting over a matrix array. Due to the memory requirements of an RM module, only a pair of X and Y routing machines could be allocated per node so that only one virtual plane of a 32x32 mesh or four virtual planes of a 16 x 16 mesh can be simulated at a time. In accordance with this memory constraint, we have decided to present results only for 16 x 16 MP1 networks in order to normalise the potential comparisons between several experiments.

A host program, written in C and using the Motif windowing system, runs on the Sun workstation and provides the interface to inject messages in the network and to collect data from the network at fixed intervals of time. The interval of measurement is 8000 cycles which is preceded by 8000 cycles to attain a steady state situation. Three simulations with different seeds for the random message generation are averaged for each result. Runs of twice this length showed differences of less than 2%, lower than the random generation variance due to the different random number generator seeds. The generation of messages is randomised in time, unless otherwise stated.

Results are obtained by increasing the network load at fixed steps until the network reaches saturation; at this point, the injection buffer (10 messages long) is unable to accept new messages. The saturation will appear first in the heaviest-loaded channels, e.g. the central ones in the random distribution. Most of the measurements shown here are made with a single-packet message of 6 bytes length, i.e. normally two header flits (8 data bits + 1 synchronisation bit each) and four data flits.

In the torus implementation simulated four chips have to be connected to implement a complete router, so in this case we only have simulated a single virtual plane of 16x16 nodes using the same DAP resources. Each subgroup of 16x16 nodes simulate the virtual plane of one of the chips.

### 4.2 Evaluated metrics

Results obtained through simulation characterise the network behaviour in terms of throughput and average message latency. Other information of interest such as channel utilisation, percentage of adaptations and buffer utilisation has also been collected for each case.

*Message latency* is defined as the time spent by a message between being injected in the network and being received by the node. In order to measure it, the cycle in which the message was put in the injection buffer is kept into the first two bytes of message data. When a message is delivered, and its reception is completed the latency is calculated as the difference between the reception and the injection cycles. Message latency includes, therefore, the injection time needed to get access to the injection channel from the interface buffer.

*Throughput* is evaluated from the number of delivered messages, and can be expressed in Kbytes/cycle; when necessary, it is related to the fraction of the network capacity resulting from the normalisation of the maximum throughput, as calculated in section 4.5. The maximum sustained throughput is considered as the highest network load achieved without flooding the

injection queue. Any attempt to generate a new message that cannot find a buffer would be a result of a working demand bigger than that supported by the network.

The other parameter which gives a clue to the network's behaviour is the *channel utilisation*. The average and highest values as a percentage of use are calculated. In each cycle, a network channel is counted as in use if its multiplexer has been assigned to one of the input channels. Both messages and the dead flits are counted. For that reason, the boundary channels have a non-zero use, although no messages go out of the networks limits.

*Injection queue* size and the *injection delay* are also of interest. If a message is generated and there is no message being injected it can be transferred within the same cycle. If not, when the previous message injection is acknowledged. Hence, the simulation model takes the simple case of no delay when the interface is available.

The goal of a network design is to come as close to 100% channel utilisation as possible. This limit is difficult to reach due to contention, fluctuation in network load and unbalanced channel utilisation.

### 4.3 Traffic patterns

In a general-purpose multicomputer the network should be able to maintain an average good performance whatever the application demands of the network. Therefore, we have evaluated the network for a set of different traffic patterns described below.

**Random traffic** in which each node sends messages to any other node with equal probability. Message generation is randomised in time. As many parallel applications exhibit a certain degree of locality they will consequently have a lower message latency as compared to this random model.

**Local traffic.** With this pattern of traffic the probability of sending a message from node to node is inversely proportional to the distance between them. Therefore, many messages will travel along short distance, decreasing the communication demands and the conflicts with the network resources.

**Permutations.** Here each node sends all of its messages to the same destination node. The pairs (source, destination) follow a permutation pattern. In our evaluation we considered three of the most widely used permutations in parallel computing, which are Perfect Shuffle, Bit Reversal and Matrix Transposition. This set of permutation patterns were simulated under two conditions, synchronous and asynchronous. In the first the network receives all the messages simultaneously when the application reaches a synchronisation point. The application will continue after the permutation has been completed. The response of the network is evaluated as the time required to deliver all the messages of a given permutation. In the second case the application processes run independently sending messages at a random time from one node to another. Synchronisation is only between a pair of processes, so the goal will be to reduce message latency in order to avoid processes being blocked by communication. In this case, we evaluate latency as a function of the load for each permutation pattern.

**Broadcast traffic.** This is a multi-point communication pattern in which a node sends a message to all the rest of nodes in the network. This can be used to control or synchronise them or to spread global information. The MP1 supports this kind of communication by a specific mechanism at the hardware level.

### 4.4 Message routing

One of our goals is to evaluate the efficiency of the communication network in routing messages and the gain due to adaptive routing. For this objective we measure the network behaviour under the same circumstances of traffic for adaptive and static routing and compare them. The MP1 routing chip supports static routing in dimensional order by a special message type that stops any attempt to find an adaptive path if the output channel is busy. We use this facility to estimate the mean message delay of the following three routing schemes:

- Oblivious: messages are routed first in the X dimension and then in the Y dimension.

- Restrictive: each message is routed first-X-then-Y or first-Y-then-X. This election being chosen randomly.
- Adaptive: messages can use any path joining source and destination.

#### 4.5 Bounded throughput

The throughput supported by the communication network is limited by the network resources. Here we establish an expression for the bound on throughput for our network which will be taken as the 100% value in estimating the network's performance. A method to establish this value, proposed by Dally for random traffic pattern, is based on the bisection bandwidth limit. Let  $q$  be the flit generation rate per node and per cycle; the number of flits that on average cross the bisection bandwidth should be

$$1/2*(N/2*q) \leq \text{Bisection bandwidth}/3$$

$$q \leq 8/(3*n)$$

In a mesh of 16x16 nodes it gives a maximum throughput of 4,067 Kbytes/s per node.

Another method to bound the network throughput, described in [19], is based on the channel utilisation and offers a more accurate analysis because it takes into account the routing strategy which has a strong impact in the success of messages in reaching their destinations. Let  $U_{X^{i,j}}$  and  $U_{Y^{i,j}}$  the percentage utilisation of the output X-link and Y-link, respectively, of node  $(i,j)$  and  $U = \max(U_{X^{i,j}}, U_{Y^{i,j}})$ . Saturation is reached when the most loaded channel is fully used, becoming the system's bottleneck. Hence the generation rate  $q$  is bounded by:

$$q \leq 1/(3*N*U)$$

With a static routing algorithm, knowing the traffic pattern, the message paths are fixed. Therefore, the link utilisation values can be easily calculated. For the dimensional order routing we know that an X-link will be used by any message injected in the same row whose destination is beyond the node in question. Similarly, a Y-link is used by any message that is injected **or turns into** a previous node of the same column. Figure 8 illustrates this scheme for oblivious and restrictive routing strategies.

##### *Oblivious*

$$U_{X^{i,j}} = (i*(n-i)*(n-j))/N(N-1)$$

$$U_{Y^{i,j}} = (i*j*(n-j))/N(N-1)$$

The most used links in a 16 by 16 mesh with virtual planes correspond to the values (8,1) and (15,8) respectively, both with the same value, giving:

$$U = \max.(U_{X^{i,j}}, U_{Y^{i,j}}) = 4/255$$

If a restrictive routing is applied an X-channel could be used by two types of messages: a message injected in the X dimension travelling in the first dimension or a message injected in the Y-dimension whose destination is in the considered node's row. With the restrictive routing mode, we expect that half of the messages to be injected into each dimension. Then:

##### *Restrictive*

$$U_{X^{i,j}} = 1/2*(i*(n-i)*n)/N(N-1)$$

$$U_{Y^{i,j}} = 1/2*(j*(n-j)*n)/N(N-1)$$

Now, the maximum values are obtained for all the link nodes at values (8,8), giving:

$$U = \max.(U_x^{ij}, U_y^{ij}) = 2/255$$

It can be seen that restrictive bounds the flit generation rate  $q$  by a value which is similar to that obtained by the former method; consequently oblivious routing in this architecture limits throughput to only 50% of its full capacity.

Finally, when the routing strategy is adaptive, a message may traverse different paths depending of the network status. The adaptive mechanism should balance the channel utilisation by using free links and alternative paths whenever a link is busy, so the most used links will be less used than in the previous two cases. It is important to remark on the effect of the splitting of resources among the four virtual networks that comprise the mesh. The utilisation factor of a channel, e.g.. east direction, of an oblivious network is split into two east channels of independent routing planes. The use of each node depends on the position of the node and is not half but follows a linear increasing/decreasing function in each plane. The restrictive routing overcome such unbalance, so that it is a better reference for the improved behaviour due to adaptations in virtual planes.

## 5. Evaluation of MP1 Networks

### 5.1 Random traffic

**Fixed message length.** Figure 9.a shows the average message latency, expressed in microseconds, as a function of the network load for oblivious, restrictive and adaptive routing schemes. A considerable gain is achieved when restrictive routing is used compared to the oblivious case. Though this routing is also deterministic, the gain is due to the balanced access of the messages to the X and Y output channels. As described above this is due to the splitting of resources between the four independent planes, the loss due to imbalance between the virtual planes is overcome with random injection into either the X or the Y dimension. Adaptation within the virtual planes also reduces message latency and increases network throughput, by exploiting, in this case, the multiple minimal paths which exist between any pair of nodes.

Figure 9.b shows the percentage of utilisation of the most loaded channel and the average utilisation of the whole network channels for each routing scheme as a function of the network load. As expected, the highest values for the critical channel correspond to oblivious routing, which reaches saturation at a load of 37% compared to the 60 or 67.5% for the other two routing methods. For all three cases the saturation point is reached when the most loaded channel supports 86% of full utilisation. Channels do not achieve full utilisation due to the time spent in the message injection process. In the case of a mesh topology with random traffic the utilisation is always much lower than the highest value due to the unbalanced use of the resources. With an adaptive routing policy, network channels are used more evenly and, as a consequence, the saturation point is reached at a higher network load.

Figure 9.c shows the average number of adaptations per message as a function of the load. Even at the highest load this is lower than one adaptation per message. The figure also shows the number of adaptations per message which occur from a buffered message due to the alternative channel becoming free first.

The figure 9.d shows the average number of buffers occupied per cycle (the 16x16 mesh has a total of 2048 buffers, i.e. eight per chip). Even at the highest load, the number of occupied buffers is not high because in each virtual plane there are empty areas where the messages rarely get blocked. With oblivious routing, the network saturates with a buffer occupation of less than 3%. As saturation is located in only a few nodes, the average buffer occupation of the injection queue is also very low per node, as can be seen in figure 9.e. When a randomised injection is used, that problem is solved and the network can handle higher loads. Basically, both blocked messages as well as the message injection are spread in the two dimensions so that the saturation point is reached at a higher load with an average buffer occupation of 9%. Finally, adaptive routing gives, logically, a lower buffer occupation, both in transit and injection.

Figure 9.f show the injection time of messages needed to pass from the injection queue to the interface buffer. This time is, as could be expected, closely related to the injection queue population, the better results being obtained when the adaptive routing is used.

**Influence of message length.** In order to observe the influence of the message length in latency, simulations have been run with different message lengths: 6 bytes, 32 bytes, 48 bytes and 96 bytes. The first two are single segment messages, the third has two segments and the last one three segments. Figure 10.a shows average message latency as a function of the load for adaptive and oblivious routing with different lengths.

Figure 10.b shows average message latency as a function of the message length for different loads considering only adaptive routing. Message length increases the transmission time and also changes the pattern of traffic in the sense that only the header packet is routed and the rest of the message follows the same path. Message latency at any load increases with the message length. However, for multi-segment messages the increment is less steep. A reason for this change can be found in the fact that there are less control decisions per node, which reduces the number of dead flits that could block a packet. Also, once the message header reaches its destination the path's channels are fully used until the whole message is transmitted.

## 5.2 Local traffic

In this experiment messages are generated randomly in time, choosing the destination node at a distance in the range 1 to 5, with the probability distribution shown in figure 11.a. The selection of one of the equidistant nodes from the source node is equi-probable. The average distance traversed by a message decreases from 10.66 with random to 2.5 with this local pattern.

Figure 11.b shows the average message latency as a function of the network load for both random and local traffic under adaptive and restrictive routing. As expected, the results show a lower latency and a highest throughput for the local patterns, this being more marked at higher loads as saturation occurs later. The late saturation point is increases network performance up to 80 % with adaptive routing and 70% with restrictive routing and the most loaded channel at saturation is only 39%. The system saturates because the injection queue overflows, due to the high message generation rate applied. Also with the local traffic pattern the chances of blocking are very reduced and therefore the gain of adaptive over restrictive routing is only noticeable at high network loads.

## 5.3 Permutation patterns

Here we analyse the behaviour of the interconnection network with traffic generated by the following permutation patterns: Perfect Shuffle (PS), Bit Reversal (BR) and Matrix Transposition (MT).

**Synchronous permutation.** Figure 12 shows the completion time for each permutation in the order described above. Figures 13.a, 13.b and 13.c show the completion time for each permutation as a function of the message length for two network sizes: 8x8 and 16x16, and for two routing strategies: oblivious and adaptive.

The MT is a clear example of a worst-case pattern for oblivious routing. All messages sent from nodes in the same row have as destination nodes in the same column. Therefore, they get blocked at the adjacent node. When the last flit of the first message of the group is delivered to the destination node, the channel is released and then all messages in a specific row advance one hop. This sequence is repeated each  $(3L+2)$  cycles until the last message of a row has been forwarded. Under these conditions, the last message to reach destination will be, for a 16x16 mesh, the message injected in node (0,15) with destination (15,0) or vice versa. The analytic value computed from this observation agrees with the simulation results.

The gain obtained using adaptive routing is quite significant for this pattern because it can exploit the multiple paths that exist for any message. On average, half of messages were injected in the Y-dimension. That means that the delay of messages in a row is halved. However, there will be also more conflicts when messages are travelling in the second dimension compared to the oblivious case, and as a consequence, messages will be delayed in that dimension.

**Asynchronous permutations.** here messages are generated randomly in time, with the destination selection dictated by the permutation pattern. Therefore the channels support, on average, the same percentage of the network load as described in the previous synchronous model. Figures 14(a),(b) and (c) show the average message latency for each of the considered permutations as a function of the network load. For the three patterns, a slight improvement in latency is achieved by using restrictive or adaptive routing at low loads. The main benefit of both strategies is seen at higher throughputs.

#### 5.4 Broadcast traffic

As already mentioned there is a network consumption mode denoted broadcast in which the source node can choose the scope of broadcast by selecting the destination address such that all nodes in the bounding rectangle receive the message's data. The labels in the figure 15 illustrate the number of cycles required to carry out a broadcast operation in a 4x4 mesh for two different models: MP1 broadcast and broadcast following a hamiltonian path.

One of the worst methods for broadcast is what we call "point-to-point" broadcasting, where the source node has to inject a new message for each destination node following a sequential procedure. The completion time of all three broadcast strategies as a function of the linear broadcast domain size can be seen in figure 16. This behaviour can be compared in the same figure with the hamiltonian strategy and with the MP1 broadcasting mode. In short, the MP1 broadcasting mechanism is very efficient, producing similar delays to those caused by a message travelling to the farthest node.

#### 5.5 Torus networks

As already discussed the message injection strategy is of major importance in order to evaluate torus networks based in MP1 chips. Three different strategies were analysed, trying to keep the injection algorithm as simple as possible. These were a fixed strategy that injected messages only in their class level and two other strategies which injected messages in their level or higher virtual levels randomly, using different weights as a function of the area of injection. The best of these strategies was then used for further simulation. This injection strategy chooses the injection level randomly among the possible options following these distribution rules:

- class HL : 1/3 to the chip level LL and 2/3 in its own level HL
- class LH : 1/3 to the chip level LL and 2/3 in its own level LH
- class LL : is a function of the network quadrant as follows:
  - NW quadrant: All to chip level LL
  - NE quadrant: 2/3 to level HL and 1/3 stay
  - SW quadrant: 2/3 to level LH and 1/3 stay
  - SE quadrant: 1/3 to level LH and 2/3 stay

In order to compare the toroidal implementation with the mesh we have simulated one plane of four MP1 chips connected in both topologies. With random traffic any virtual plane manages the same load so that for this pattern the results are representative of the whole network characteristics. The four chips share a single interface, which sequentially injects messages to any of the 16 virtual planes.

Figure 17 shows the average latency as a function of the normalised load. The torus network exhibits an average message latency lower than the mesh at any level of network load. This is due to the lower diameter and average distance of the torus compared to the mesh topology and because the network load is more balanced in the torus than in the mesh network. It also has a higher maximum throughput compared to the mesh performance. The lower performance of the four chip mesh network compared to the single chip network is due to the contention in the injection and reception channels of the interface.

## 6. Summary

In this paper we carry out an evaluation study of the first version of the Mad Postman Routing Chip (MP1). This message router has been designed, implemented and tested by the Computer Systems Research Group at the University of Surrey and it is the first step in the development

of fine-grain scalable parallel computers. The issues that have been addressed are the development of efficient hardware message-passing managers.

The main contribution of this paper is the evaluation of the capabilities of bidimensional networks based on MP1 chips. To achieve this goal, we have observed the network responses running under several realistic load conditions. This analysis accurately reflects the real behaviour of a physically constructed subsystem and has been found invaluable in improving the cost/performance trade-off of successive refinements when facing the design of the internode communication mechanisms for massively parallel computers.

## 7. Acknowledgements

We would like to acknowledge the following for their inputs in the MP1 design process:

Dr P. Miller  
Dr J. Yantchev  
Dr C. Huang  
Mr C. Lee and  
Mr I. McNally.

This work has been funded by SERC, under grant GR/G 69680 with partial support from the Spanish CICYT under contract no: TIC-91 0389.

## 8. References

- [1] Bell, G (1992) Ultracomputers. A teraflop before its time, *CACM*, **35** (8), pp27-47.
- [2] Leirson, C. et. al., (1992) The network architecture of the connection machine CM5, *Proc.SPAA 92* pp272-285.
- [3] Meiko (1993) Computing Surface 2 overview documentation set, Meiko Ltd, Bristol.
- [4] Hockney, R. W. (1989) *Parallel Computers 2*, Adam Hilger Ltd.
- [5] Lenoski, D. et. al. (1992) The Stanford DASH multiprocessor. *IEEE Computer*, march 1992, pp63-79.
- [6] Agarwal, A (1992) Performance tradeoffs in multi-thread processors, *IEEE Trans. Parallel and Distributed Systems*, **3**, pp525-539.
- [7] Jesshope, C. R. (1992) The f-code abstract machine and its implementation, *Proc. 6th Annual European Computer Conference on Computer Systems and Software*, IEEE Computer Society Press, pp169-174.
- [8] Seitz, C. L. (1985) The cosmic cube, *CACM*, **28**(1), pp22-33.
- [9] Zopette, G. (1992) The power of paralelism, *IEEE Spectrum*. Sept, pp28-33.
- [10] Dally, W. J. (1992) The message driven processor: a multicomputer processing node with efficient mechanisms, *IEEE Micro*, April, pp23-39.
- [11] Kermani, P. and Kleinrock, L. (1979) Virtual cut-through: a new computer communication switching technique, *Computer networks*, **3**, pp267-286.
- [12] Dally, W. J. and Seitz, C. L. (1986) Deadlock free message routing in multiprocessor interconnection networks, *IEEE Trans .Comput.* **C-36**, pp547-553.

- [13] Yantchev, J.T. and Jesshope, C.R. (1989) Adaptive low-latency deadlock-free packet routing for networks of processors, *IEE Proc. E*, **136**, pp178-186.
- [14] dally, W. J. and Seitz, C. L. (1986) The torus routing chip, *Distributed Computing*, **1**, pp187-196.
- [15] Linder, D. H. and Harden, J. C. (1990) An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes, *IEEE Trans. Comput.*, **C-40**, pp2-12.
- [16] Miller, P. R. (1991) Efficient communications for fine-grain distributed computers, Ph. D. Thesis, University of Southampton.
- [17] Jesshope, C. R. and Izu, C. (1993) The MP1 network chip, *proc. Euromicro workshop on Parallel and Distributed processing*, Gran canaria, pp338-348, IEEE Computer Society press.
- [18] Computer Systems Research Group (1992) *The Mad-postman network chip MPI reference manual*, Internal report, Dept of Electronics and Elec. Eng., University of Surrey.
- [19] Reed, D.A. and Fujimoto, R. M. (1987) *Multicomputer networks: message based parallel processing*, The MIT press.

## Figure captions

Figure 1. Architecture of the MP1 chip. All channels shown are three bit wide channels capable of passing a byte of data + synchronisation bit in three channel cycles (120ns).

Figure 2. The structure and connections of a single plane node.

Figure 3. The structure of a multiple segment message.

Figure 4. How to unwrap a torus using multiple levels of virtual networks.

Figure 5. How to reduce the resources required in a multi-level torus network. Only those nodes shaded in level 2 will ever reach level 1, and no message from level 1 will ever reach the shaded area of level 0.

Figure 6. Multi-level torus network using the MP1 chip. Four chips are required as each chip contains a single node address register. The scheme of figure 5 could only be used if an address register were provided for each plane in the MP1 chip.

Figure 7. A photograph of the prototype 16 node MP1 network. Processor boards containing an interface chip are mounted orthogonally to this "backplane" board by 20 pin connectors (on reverse side of the board).

Figure 8. An illustration of how loading calculations are performed for oblivious and restrictive routing strategies.

Figure 9. Simulation results for uniform random traffic for messages of 6 bytes in a 16x16 MP1 mesh network.

- (a) latency versus throughput.
- (b) Channel utilisation vs throughput.
- (c) Number of adaptations versus throughput.
- (d) Number of occupied buffers versus throughput.
- (e) Number of occupied injection buffers versus throughput.
- (f) Injection latency versus throughput.

Figure 10. Variation of latency with message length drawn as a function of throughput (a) and message length(b).

Figure 11. Simulation of local traffic conditions.

(a) Message distribution as a function of number of hops, compared to uniform random traffic.

(b) Latency against throughput for local and uniform random traffic.

Figure 12. Synchronous permutation time for different routing strategies. Permutations are from left to right, perfect shuffle, bit reversal and matrix transpose.

Figure 13. Variation of synchronous permutation time against message length, for perfect shuffle (a), bit reversal (b) and matrix transpose (c).

Figure 14. latency of asynchronous permutation traffic against for perfect shuffle (a), bit reversal (b) and matrix transpose (c).

Figure 15. Comparison of MP1 broadcast tree with that obtained using a hamiltonian path.

Figure 16. Time to complete a broadcast to a square area against linear dimension for MPI broadcast, hamiltonian path and for point-to-point. The hamiltonian result is analytical and assumes the same delivery latency in the MPI hardware broadcast.

Figure 17. Latency against throughput for a 16x16 Mesh network against a 16x16 torroidal network using 4 Mp1 chips at each node.