Network Fault-Tolerance with Interval Routing Devices*

Johan Vounckx, Geert Deconinck, Rudi Cuyvers, R. Lauwereins**, J. A. Peperstraete

Katholieke Universiteit Leuven,
Electrotechnical Department, Group ESAT-ACCA
Kardinaal Mercierlaan 94
3001 Heverlee, Belgium

tel.: +32-(0)16-22 09 31 / fax.: +32-(0)16-22 18 55

email: Johan.Vounckx@esat.kuleuven.ac.be

Abstract Massively Parallel Computers are built out of millions of devices, making the overall system's dependability too low. Fault-tolerant measures have thus to be included. In this paper we will address *network fault-tolerance*. More specifically, a routing algorithm is developed which is able to isolate faulty units, entire nodes as well as single links, and to reroute messages to spares. The proposed method, based on multi-interval routing, is near-optimal and requires only few extra intervals.

keywords: massively parallel computer, fault-tolerance, multiple interval routing, networks

1. Introduction

To meet the increasing demand for computational power one needs MPC (Massively Parallel Computers). A first requirement of MPC is that they should be scalable. Not only more processors should be used, the performance of the communication network should also increase. Delays should be very small. Switching methods, offering small delays are wormholing [2] and virtual cut-through [9]. Scalability also requires that the amount of routing information is independent of the number of processors. One such compact routing technique is interval routing [3], [4].

MPC are built out of millions of devices, making the overall system's dependability unacceptably low. A second requirement of MPC's is thus that fault-tolerant measures must be included. The presence of failures causes problems for the compact routing algorithms. Compact routing tables, allowing optimal routing (shortest path) are generally based on a regular structure. For optimal routing, information is needed. In a regular network, this information can be implicit, thus resulting in less information actually to store. In the presence of faults, this implicit information is no longer valid and has to be replaced by new, explicit, information. In [8], fault-tolerance for interval routing was studied for a limited class of networks. A routing algorithm is developed which is able to isolate faulty units, entire nodes as well as single links, and to reroute messages to spares.

* Partly supported by Esprit project 6731 (FTMPS) and IUAP-50

** Senior Research Associate of the Belgian National Science Foundation

In the second section of the paper we will describe the machine our algorithms are to be run on. We suppose the routing device to support multi-interval routing [3], [5]. We will briefly explain some necessary concepts of multi-interval routing and its limitations. A description of fault-free routing for this system will be given, since it forms the basis for the fault-tolerance algorithms.
Next we will consider the faults that may occur. We will then also state the general ideas about how to overcome these failures. In the following sections we will explain the algorithms we have developed to recover from these failures.
Finally, a conclusion will state the trade-off between the different proposed algorithms.

2. The machine

Our routing algorithms are developed for a communication network that consists of a 2D- or 3D-grid. Possibly the nodes themselves are complex structures, containing many processors, some of which may be used as spares, connected by some network. An example of such a machine is the Parsytec GC [7]. This paper will address routing in the grid.

The routing devices used in the communication network are based on interval labelling. More specifically, they support multi-interval routing, such as the C104 router of Inmos [6]. We will first explain some concepts of interval routing.

2.1. Interval routing

Interval routing was announced in [4]. In [1] it was shown that all networks support interval routing.

The principle of interval routing (figure 1) is quite simple. First, processors are uniquely labelled. At each node a table of intervals of these labels is formed. At each intermediate node, the message is transmitted over the link corresponding to the interval its header (destination node's label or address) belongs to. Each interval is associated to a label, indicating the start of the interval.

The assigning of labels is a centralised task. No algorithm is known by the authors that allows a distributed labelling and achieves global consistency. Not every labelling is valid [1]. In [3] a depth-first method is presented to label the processors of any network in such a way that each link corresponds to only one interval. This labelling scheme however

results in not optimal routing. In fact, mono-interval routing is only optimal for some topologies [1]. Some optimal labelling schemes can be found in [11].

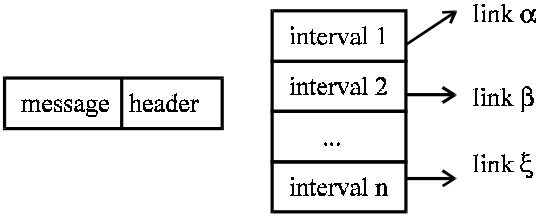


Figure 1: Interval labelling. The message is output on a link, according to the interval the header belongs to.

Therefore we might consider to use more than one interval per link. This is also called multiple label routing, since links are now allowed to have more intervals, and thus more labels.

2.2. Fault-free routing

2.2.1. Routing and labelling

We will start by explaining the routing mechanism in the fault-free case. Since we want to handle faults as locally as possible, fault-free routing forms the basis for the fault-tolerant routing.

We assume the following conventions (figure 2):

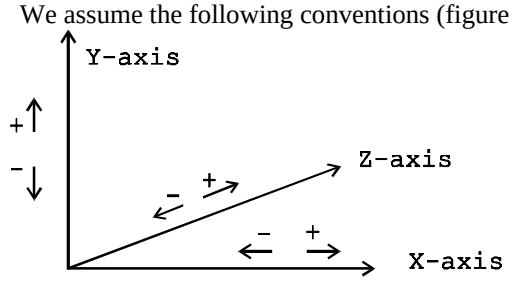


Figure 2: Conventions for the grid-axes

An optimal routing algorithm can be:

- messages for nodes lying in a previous/next plane are sent by the links in the -Z/+Z direction,
- messages for nodes belonging to rows above/below this node are sent on links in the +Y/-Y direction,
- messages for nodes of the same row, but lying at the left/ right, are sent by links in the -X/+X direction,
- messages for this node reached their destination.

This can easily be implemented in interval tables, using one interval for each direction. We use (x,y,z) as coordinates in the 3D-grid, and A, B and C as the number of nodes in each dimension. Nodes can then uniquely be numbered by A*B*z+A*y+x. We then get the following interval tables (messages for nodes having a label less than the boundary indicated in the left column are output according to the right column):

A*B*z	to front (-Z)
A*B*z+A*y	downwards (-Y)
A*B*z+A*y+x	to left (-X)
A*B*z+A*y+x+1	inside node
A*B*z+A*(y+1)	to right (+X)
A*B*(z+1)	upwards (+Y)
A*B*C	to back (+Z)

Messages will first be routed to the appropriate plane, then to the appropriate row. Ultimately, they move towards the correct node.

2.2.2. Definitions

We conclude this section with some definitions. The X-direction will also be referred to as the horizontal direction, the Y-direction as the vertical one. All the nodes sharing the same y-coordinate are referred to as a *vertical line*. All the nodes sharing the same x-coordinate are referred to as a *horizontal line*. Nodes sharing the same z-coordinate are form a *plane*.

$I_{u,l}$ = {v | messages for v are output on link l at node u}
Suppose a = (x_a,y_a,z_a)
$R_a = \{v=(x_v,y_v,z_v) \mid z_v=z_a,\ y_v=y_a,\ x_v>x_a\} = I_{a,+X}$
$L_a = \{v=(x_v,y_v,z_v) \mid z_v=z_a,\ y_v=y_a,\ x_v<x_a\} = I_{a,-X}$
$U_a = \{v=(x_v,y_v,z_v) \mid z_v=z_a,\ y_v>y_a\} = I_{a,+Y}$
$D_a = \{v=(x_v,y_v,z_v) \mid z_v=z_a,\ y_v<y_a\} = I_{a,-Y}$
$F_a = \{v=(x_v,y_v,z_v) \mid z_v<z_a\} = I_{a,-Z}$
$A_a = \{v=(x_v,y_v,z_v) \mid z_v>z_a\} = I_{a,+Z}$

In stead of defining $I_{u,l}$ relative to a single node, we can also define $I_{region,l}$, relative to a convex region of nodes. These definitions are straightforward extensions of the previous ones. The subscript indicating which region is meant may be omitted.

2.3. Faults

A massively parallel computer consists mainly of three types of components. At first there are the nodes that can be subdivided into a computational unit, possibly consisting of several interconnected *processors*, and a communication unit or *routing device*, connecting the node to the communication network. The communication network consists of these routing devices and *links*.

When a processor fails, it is no longer usable and should thus be replaced by a spare one, requiring isolation of the faulty processor and rerouting to the spare. This is straightforward if a spare in the same node is used. If no spare in the same node can be found, one might consider the use of a spare in a neighbouring node. Routing will become much more difficult. When the complete computational unit of a node fails, that node has to be replaced by a spare. The routing device can still be used. In case of a routing switch failure the entire node has no longer access to the communication network. This node is no longer usable and a spare one has to be found, requiring isolation of the failed one, and rerouting of messages to the spare. In case of a link failure, messages have to be routed around that link.

Fault-free routing thus has two major tasks: Messages have to be routed around faulty components and to spares. In the next section we will deal with the isolation aspect of fault-free routing. The following section handles routing to spares.

3. Routing around faulty entities

For reasons of scalability, we use a local approach to overcome failures. Of course we have to take care to

achieve global consistency through the local deroutings. No loops or conflicts may arise.

We propose a method where the extra amount of intervals is dependent on the number of faults but not on the complexity of the network. The resulting routing will be near optimal. The paths will only be slightly longer than in the fault-free case and the number of extra hops does not depend on the size of the network.

3.1. A single failed link

Link failures prevent messages for nodes of a certain interval to be delivered in a normal way. An alternative path must be found. We will use local rerouting, i.e. the messages are locally routed around the failed link. This will be possible with a limited change to the routing tables.

3.1.1. Link faults in the Z/ Y-direction can be handled by passing the messages first on a Y, X/ X-link. This asks no extra intervals to be used (the same interval is simply output on an other link) and only the nodes connected by the failed links have to be adapted. It is even possible to reduce the number of intervals by choosing the outgoing link of an adjacent interval. Notice that several reroutings are possible: When a link in the Z-direction fails, one of 4 links (+/- Y, +/- X) may be chosen. When a link in the Y-direction fails, one of 2 links (+/- X) may be chosen. The two affected nodes do not have to choose the same replacing link. This is even preferable for load-balancing.

We consider only one case, e.g. a vertical link failure handled by sending the messages over the +X link, to further explain this rerouting. The other cases are completely analogous.

Consider a vertical link failure between nodes a and c (figure 3). We start by considering the situation at node a. Messages that should use the failed link are destined for all nodes lying under node a (encircled by the shaded box). These messages can only come from the Y-direction above a, from the node a itself or from the Z-direction. They are first passed through node b. No further action is needed for these messages. They will normally be routed on the vertical line until the correct horizontal line is reached. There they pass to that horizontal line to be routed to their destination node. Paths thus possibly contain two extra hops.

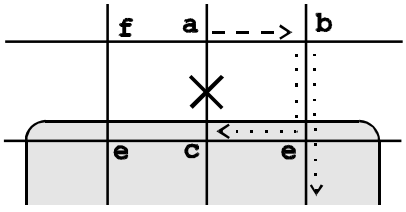


Figure 3: A vertical link failure is handled by sending the messages for D_a first to b.

Of course at node c, analogous changes are necessary. This routing scheme is valid, no loops can occur: To prove this, it is sufficient to show that no loops occur for the changed paths, i.e. messages for nodes belonging to D_a, could cause loops. At b, the messages for D_a are routed by the fault-free routing network correctly to D_a and will not return to either a or b. The interval tables for node a and c become:

node a:

A*B*z	-Z
A*B*z+A*y	+X
A*B*z+A*y+x	-X
A*B*z+A*y+x+1	node
A*B*z+A*(y+1)	+X
A*B*(z+1)	+Y
A*B*C	+Z

node c:

A*B*z	-Z
A*B*z+A*y	-Y
A*B*z+A*y+x	-X
A*B*z+A*y+x+1	node
A*B*z+A*(y+1)	+X
A*B*(z+1)	+X
A*B*C	+Z

3.1.2. Link faults in the X-direction require changes in the two nodes connected by the failed link and in two neighbouring nodes. The neighbouring nodes are required to put the messages on a horizontal link again.

Suppose the link between nodes a and d (Figure 4) fails. Messages for R_a can no longer be routed on link (a,d). We can simply reroute these messages by sending them upwards to b. This is shown by the dashed arrow. At b however, messages for R_a, are sent back to a.

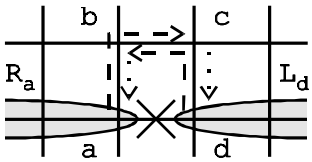


Figure 4: Rerouting to handle horizontal link failures.

Thus, also at b, intervals have to be changed. We have to introduce an extra interval at b, corresponding to R_a and send these messages to c. Notice that the other intervals remain unchanged. At c, no further changes are necessary. Messages for R_a are routed downwards to d. This is shown by the dotted arrow. The messages thus jump around the failed link. They have 2 extra hops to pass.

Of course, at d and c, equivalent changes are necessary for the messages destined for L_d. Of course, messages can also be derouted in the -Y direction also. In fact, nodes a and d could reroute the messages for R_a and L_d in opposite Y direction.

Not only the Y-direction, but also the Z-direction can be used to reroute messages for R_a and L_d. Up to four rerouting paths can thus be formed. Again, the two affected nodes do not have to choose the same direction to replace the failed link.

Also here no loops are possible. The messages on the changed paths just jump around the failed link and

are then normally routed. No message will ever return to the node before the link.

The extra number of intervals varies. At the two nodes that are connected to the failed link, the messages only have to be redirected. No extra interval is needed. It is even possible to use one interval less, if we use the same outgoing link as an adjacent interval. The neighbouring nodes that have to change their interval tables need one or two extra intervals, depending on the position of R_a and L_d in their intervals: If Ra/Ld are adjacent to an existing interval on the neighbouring nodes, one extra interval is needed, otherwise two.

If we put the rerouting via the +Y direction into interval tables, we get for a, b, c and d:

node b:

$A*B*z_b$	-Z
$A*B*z_b+A*y_a$	-Y
$A*B*z_b+A*y_a+x_a$	-X
$A*B*z_b+A*y_b$	-Y
$A*B*z_b+A*y_b+x_b$	-X
$A*B*z_a+A*y_b+x_b+1$	node
$A*B*z_a+A*(y_b+1)$	+X
$A*B*(z_b+1)$	+Y
$A*B*C$	+Z

node c:

$A*B*z_b$	-Z
$A*B*z_b+A*y_a+x_a$	-Y
$A*B*z_b+A*y_b$	+X
$A*B*z_b+A*y_b+x_b$	-X
$A*B*z_a+A*y_b+x_b+1$	node
$A*B*z_a+A*(y_b+1)$	+X
$A*B*(z_b+1)$	+Y
$A*B*C$	+Z

node a:

$A*B*z_a$	-Z
$A*B*z_a+A*y_a$	-Y
$A*B*z_a+A*y_a+x_a$	-X
$A*B*z_a+A*y_a+x_a+1$	node
$A*B*z_a+A*(y_a+1)$	+Y
$A*B*(z_a+1)$	+Y
$A*B*C$	+Z

node d:

$A*B*z_d$	-Z
$A*B*z_d+A*y_d$	-Y
$A*B*z_d+A*y_d+x_d$	+Y
$A*B*z_d+A*y_d+x_d+1$	node
$A*B*z_d+A*(y_d+1)$	+X
$A*B*(z_d+1)$	+Y
$A*B*C$	+Z

3.1.3. Link faults in the Y-direction can also be solved by routing the messages belonging to U_{node} or D_{node} through links in the Z-direction. Then the same changes, this time in the Z-Y plane, are needed as for a horizontal link failure. The extra number of intervals and hops also equals the above-mentioned solution.

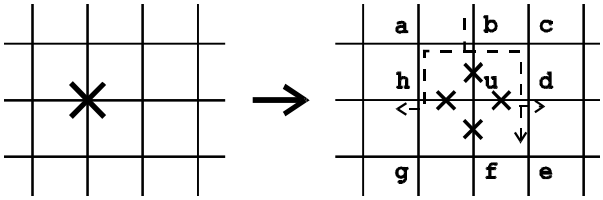
3.1.4. Splitting: We will now give some possible ways to split intervals while derouting the messages. As we will see further, this splitting technique is used to overcome multiple failures. Of course, this technique results in more optimal routing and a better load-balancing.

When bypassing through a link in the X-direction to overcome Z-/Y-link failures, we can split the interval $I_{u,link}$ in $L_{u'}$ (or $R_{u'}$) (u' = node, connected to u through the failed link) and $I_{u,link} \setminus L_{u'}$ (or $R_{u'}$), instead of sending the whole interval over the same link. Messages for the first interval are sent through the -(+)X-direction, messages for the second interval are sent through the +(-)X-direction. This approach requires one extra interval at the two affected nodes.

When bypassing through a link in the Y-direction to overcome Z-link failures, we can split the interval $I_{u,link}$ in $D_{u'}$ (or $U_{u'}$) (u' = node, connected to u through the failed link) and $I_{u,link} \setminus D_{u'}$ (or $U_{u'}$), instead of sending the whole interval over the same link. Messages for the first interval are sent through the -(+)Y-direction, messages for the second interval are sent through the +(-)Y-direction. This approach requires one extra interval at the two affected nodes.

3.2. A single failed node

Routing tables in the nodes around the failed node have to be adapted. A single node failure is in fact, a superposition of link failures. This is shown in figure 5.



a) b)

Figure 5: A node failure can be handled as a superposition of link failures.

An individual single link failure has four possible deroutings to overcome the failure. One can not use an arbitrary combination of these possible new paths in order to be consistent and to avoid loops.

Messages intending to use the failed Z-direction can be rerouted by sending them on one of the four links in the Y/ X-direction at nodes (i_u,j_u,z_{u-1}) and (i_u,j_u,z_{u+1}) (figure 5b). If their destination node does not belong to the same plane as u, these messages are transmitted to an other plane and there they will be routed properly. If, on the other hand, the destination nodes belong to the same plane as u, and if at b, f, d and h, the necessary measures (to deal with the vertical and horizontal link failures at these nodes) are taken, this results in valid routing.

Messages intending to use the failed Y-direction can be derouted by passing them either over a link in the Z-direction or over a link in the X-direction. We will not discuss the first case here, since this is completely equivalent to the recovery from horizontal link failures, mentioned in section 3.1. The latter case

offers two possibilities: whether using only one of the two horizontal links, whether using both (figure 5b).

We start by discussing the possibility that all messages for D_b are derouted on the same link, e.g. b-c. One thus need to have a path b→c→d(→...) for messages belonging to D_b. This only asks derouting at b, since at c these messages are already routed downwards (fault-free routing). On the other hand, we need a path b→c→d→e→f→g→h for messages destined to L_u. This implies that at f, these messages necessarily are output on link f-g. At node g, the messages for L_u are normally transmitted to node h. If we also at f want to restrict to one interval for U_f, the chosen link has to be f-g. If at d, the messages for Lu are properly derouted, messages derouted due to the vertical link failures are all routed properly to their destination.

Next we discuss the possibility that D_b is split into L_u and the rest. Messages for L_u are routed to a. There they are normally routed to h. The messages for $D_b \backslash L_u$ are routed to c. There these messages are properly routed downwards. At f, analogous measures are necessary.

Messages intending to use the failed X-direction have the same choice as for a single link failure. They can bypass the failed link via the Y-direction or via the Z-direction. We start with discussing the former case. Consider node h. Messages for R_u need to be derouted. There are no restrictions when passing via a vertical link, since at a or g no changes have been made to the interval tables yet. No conflicting situation can thus arise. At b or f, on the other hand, changes have already been incorporated. These changes must not be undone, since then consistency can not be guaranteed. We must thus choose an alternative path that does not cause any conflicts with previous changes. This limits the choice of the alternative paths. It is important to notice that if, for recovery of the vertical link failures, D_b and U_f have been divided into two parts, $D_b \backslash L_u$ and L_u, and $U_f \backslash R_u$ and R_u, no conflict situation arises at nodes b and f.

The horizontal link failures can also be bypassed through the Z-direction. We have two options: derouting from node h to node (x_h,y_h,z_{h-1}) or to node (x_h,y_h,z_{h+1}). The interval tables need to be adapted at node $(x_h,y_h,z_{h\pm1})$ to route the messages for R_u to $(x_u,y_u,z_{u\pm1}) = (x_{h+1},y_{h+1},z_{h\pm1})$. At these nodes conflicts may arise when the measures to overcome the link failure in the Z-direction route the messages destined to nodes belonging to the same plane as the failed node u (including R_u) to node h. If one takes care to avoid these situations, the messages for R_u are properly routed to $(x_d,y_{d+1},z_{d\pm1})$. From there, the messages are routed to d as in the fault-free case.

Just as for the vertical links, at $(x_u,y_u,z_{u\pm1})$, A_u or F_u can be split into two intervals, A_u and $A_u \backslash L_u$ or R_u and F_u and $F_u \backslash R_u$ or L_u. No conflicting situation with recovery from horizontal link failures can then arise.

As explained, several solutions are possible to reroute the messages in case of a node failure. We give one of them. First we indicate the necessary changes.

node	changes	extra intervals
a	none	0
b	$L_u\leftarrow$, $D_b \backslash L_u \rightarrow$	0
c	$L_u\leftarrow$	2
d	$L_d\uparrow$	0
e	none	0
f	$R_u\rightarrow$, $U_f \backslash R_u\leftarrow$	2
g	$R_u\rightarrow$	2
h	$R_h\downarrow$	0
(i_u,j_u,z_{u-1})	$A_u\uparrow$	-1
(i_u,j_u,z_{u+1})	$F_u\downarrow$	-1

The intervaltables of this example are:

node	b
$A*B*z_b$	-Z
$A*B*z_b+A*y_u+x_u$	+X
$A*B*z_b+A*j_b+i_b$	-X
$A*B*z_b+A*j_b+i_b+1$	node
$A*B*z_b+A*(j_b+1)$	+X
$A*B*(z_b+1)$	+Y
A*B*C	+Z

node	c
$A*B*z_c$	-Z
$A*B*z_c+A*y_u$	-Y
$A*B*z_c+A*y_u+x_u$	-X
$A*B*z_c+A*y_c$	-Y
$A*B*z_c+A*y_c+x_c$	-X
$A*B*z_c+A*y_c+x_c+1$	node
$A*B*z_c+A*(y_c+1)$	+X
$A*B*(z_c+1)$	+Y
A*B*C	+Z

node	g
$A*B*z_g$	-Z
$A*B*z_g+A*y_g$	-Y
$A*B*z_g+A*y_g+x_g$	-X
$A*B*z_g+A*y_g+x_g+1$	node
$A*B*z_g+A*(y_g+1)$	+X
$A*B*z_g+A*y_u+x_u$	+Y
$A*B*z_g+A*(y_u+1)$	+X
$A*B*(z_g+1)$	+Y
A*B*C	+Z

node	h
$A*B*z_h$	-Z
$A*B*z_h+A*j_h$	-Y
$A*B*z_h+A*j_h+i_h$	-X
$A*B*z_h+A*j_h+i_h+1$	node
$A*B*z_h+A*(j_h+1)$	-Y
$A*B*(z_h+1)$	+Y
A*B*C	+Z

node (i_u,j_u,z_{u-1})	
A*B*z	-Z
A*B*z+A*j	-Y
A*B*z+A*j+i	-X
A*B*z+A*j+i+1	node
A*B*z+A*(j+1)	+X
A*B*C	+Y

node	d
A*B*z_d	-Z
A*B*z_d+A*y_d	-Y
A*B*z_d+A*y_d+x_d	+Y
A*B*z_d+A*y_d+x_d+1	node
A*B*z_d+A*(y_d+1)	+X
A*B*(z_d+1)	+Y
A*B*C	+Z

node	f
A*B*z_f	-Z
A*B*z_f+A*y_f	-Y
A*B*z_f+A*y_f+x_f	-X
A*B*z_f+A*y_f+x_f+1	node
A*B*z_f+A*(y_f+1)	+X
A*B*z_f+A*j_u+x_f	-X
A*B*z_f+A*(y_u+1)	+X
A*B*(z_f+1)	-Y
A*B*C	+Z

node (i_u,j_u,z_{u+1})	
A*B*z	-Y
A*B*z+A*j+i	-X
A*B*z+A*j+i+1	node
A*B*z+A*(j+1)	+X
A*B*(z+1)	+Y
A*B*C	+Z

3.3. Convex region failures

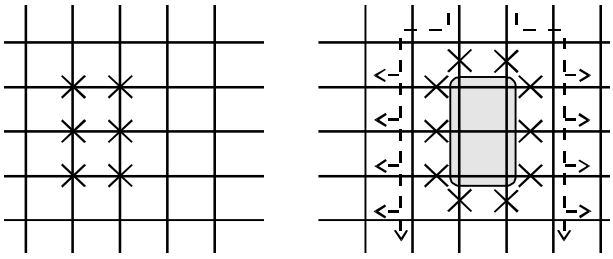


Figure 6: A convex region fault and its equivalent failed links.

A whole convex region (figure 6), consisting of multiple nodes, is considered to be faulty. Analogously to a single node failure, we adapt the surrounding nodes so that messages are routed around the failed system part. For this so-called "border routing" extra intervals are required, at most twice the number of the nodes in the vertical dimension of the failed region.

At first sight we can convert the region failure to link failures at its borders. However, superposition of the interval-adaptations required by each individual link fault can lead to inconsistency, just as for the single node failure. The solution to this problem is based on the splitting of intervals. This technique is further explained in section 3.3.1.

"Cycle routing" is an alternative solution. In a first step messages are routed to the correct Z-plane, in the same way as the above mentioned border routing. Next messages are routed in a cycle around the failed region. Compared to border routing, much less intervals are required. The routing is however less optimal. We explain this technique further in section 3.3.2.

3.3.1. Border routing

Suppose the upper left node of the convex region has coordinates (x_{ul},y_{ul},z_{ul}) and the lower right node (x_{lr},y_{lr},z_{lr}), $x_{ul}<x_{lr}$, $y_{ul}>y_{lr}$ and $z_{ul}<z_{lr}$. We call the convex region CR. We consider first the nodes between (x_{ul},y_{ul},z_{ul-1}) and (x_{lr},y_{lr},z_{ul-1}). We call these set of nodes F. These nodes lie in the plane, just before the failed convex region. Messages will fail to use the plus Z-link on these nodes. Thus, we have to reroute the interval A_{node}. Fortunately, this interval is the same on all nodes of F. We thus reroute messages for A_{node} to B_F = {(x_{ul-1}/x_{lr+1},y_{lr-1}..y_{ul+1},z_{ul-1}) ∪ (x_{ul-1}..x_{lr+1}, y_{ul+1}/y_{lr-1}, z_{ul-1})}, the border of F (figure 7). Arrived at B_F, the messages will move towards their destination plane. If further routing is guaranteed to be consistent, all messages are properly delivered.

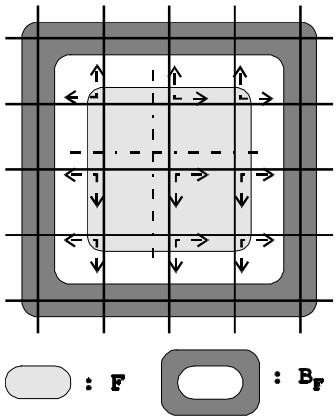


Figure 7: Rerouting messages in the plane for a failed convex region.

We divide F into four equal regions. The left upper region may reroute messages for A_{node} on -X or +Y-links, the right upper region on +X or +Y-links, the left lower region on -X or -Y -links and the right lower region on +X or -Y-links (figure 7).

Of course, the nodes between (x_{ul},y_{ul},z_{lr+1}) and (x_{lr},y_{lr},z_{lr+1}), lying in the plane just after the failed convex region, should be treated in the same way. We call these nodes A.

We next have a look at the nodes just above the failed convex region, i.e. between nodes (x_{ul},y_{ul+1},z_{ul}) and (x_{lr},y_{ul+1}, z_{lr}). We call these set of nodes U = {U_Z|z_{ul}≤z≤z_{lr}}. We now only have to consider plane by plane, since the above measures guaranteed delivery at

the correct plane. Messages will no longer be able to use the -Y-link at nodes of U_z to arrive at the correct horizontal link. We thus have to provide alternative paths. We do not allow the messages to leave the plane, since then loops can occur. The messages will have to move to the left or the right border of U, which we call $RL_U \subset B_U$, the border of U. From there they will be routed to the correct horizontal line. This results in an overall consistent routing if the messages can go to the correct node on that horizontal line. Therefore it is necessary that the destination node lies on the same side of the failed convex region as the current intermediate node. This determines at nodes of U when to route messages to the left and when to the right border.

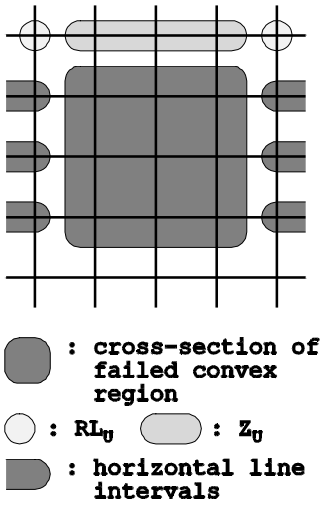


Figure 8: Rerouting messages in the plane above a failed convex region.

We explain this further with figure 8. At each node of U, an interval is necessary for each horizontal line, left from the convex region and for each horizontal line, right from the convex region. The outgoing link should of course be set according to the side of the convex region the target horizontal line is situated on. The nodes of RL_U and the nodes under RL_U do not have to change. The messages will be normally routed to the correct horizontal line.

Of course, the same adaptations are needed for the nodes under the failed convex region, called D.

We now adapt the nodes just at the left of the failed convex region, i.e. the nodes between $(x_{ul-1},y_{ul}, z_{ul})$ and (x_{ul-1},y_{lr},z_{lr}). We call these set of nodes L. Analogously, the nodes between (x_{lr+1},y_{ul},z_{ul}) and (x_{lr+1},y_{lr},z_{lr}) form a set called R. These nodes lie just at the right of the failed region. We divide L in sets of nodes belonging to the same plane: $L = \cup L_z$, Lz ={nodes v| v∈L & z_v=z}. At the nodes of L, messages will no longer be able to use the +X-links. We thus have to deroute these messages. As explained in the paragraph on single link failures this is possible via links in the Y- and Z-direction.

We start by considering a bypass in the Y-direction. Consider a node a = (x_a,y_a,z_a) of L. The messages for R_a(\CR) can not be routed through the +X-link. An alternative path is necessary. We choose to route these messages on the +Y-link. So they arrive at node (x_a,y_{a+1},z_a). Normally they would be routed down again to a. We thus need to adapt the interval tables on (x_a,y_{a+1},z_a) by changing the outgoing link for R_a(\RC). In the same way, we need to assign a new outgoing link for R_a(\RC) on all nodes between a and (x_a,y_a,z_a). On node (x_a,y_{ul+1},z_a) the messages do no longer have to be routed in the +Y-direction, but in the plus X-direction. Doing so, the messages enter U.

In U, they are correctly routed to the other side of the convex region and then to the correct horizontal line.

We have to repeat this step for each node of L_z, requiring $2(y_{ul}-y_a)$ extra intervals on node a. We explain this in figure 9.

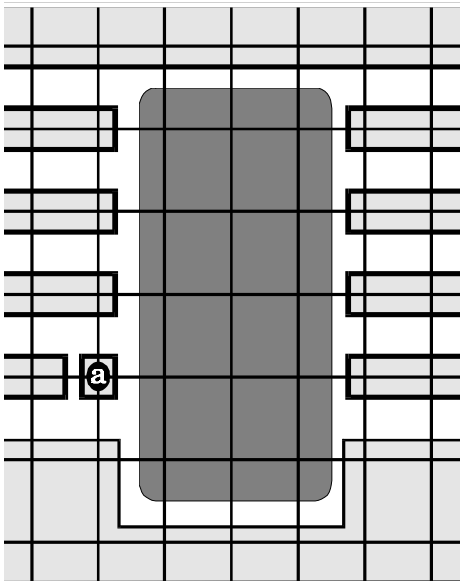


Figure 9: The hatching indicates (a cross-section of) the failed region, the light grey areas the necessary intervals at some node a. Node a needs $2(y_{ul}-y_a)$ extra intervals compared to the fault-free case.

It is naturally possible also to use the -Y-direction. Combination of both +Y and -Y-direction in the same L_z, increases bandwidth and reduces delays. One can reroute messages for R_{node}(\CR) on +Y-links if $y_a>(y_{ul}-y_{lr})/2$ and on minus Y-links otherwise (figure 10). The grey arrow indicates however that this does not result in local shortest path routing for all messages. Messages for R_a(\CR) will first move to a and then will be routed to R_a(\CR) over a (locally) shortest path.

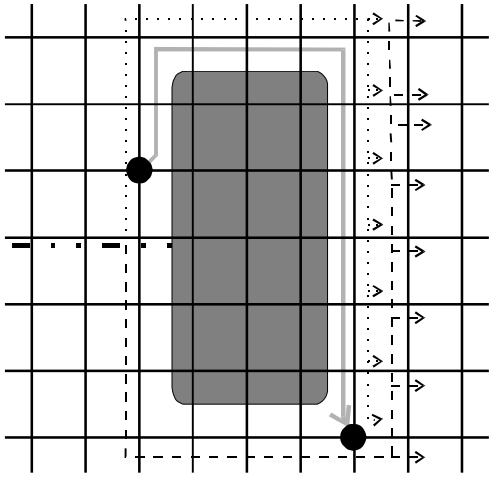


Figure 10: The hatching indicates the failed convex region. The dashed and dotted lines indicate the rerouting of messages on ±Y-links to recover from convex region failures. The full grey line indicates a possible rerouting, which is clearly not optimal.

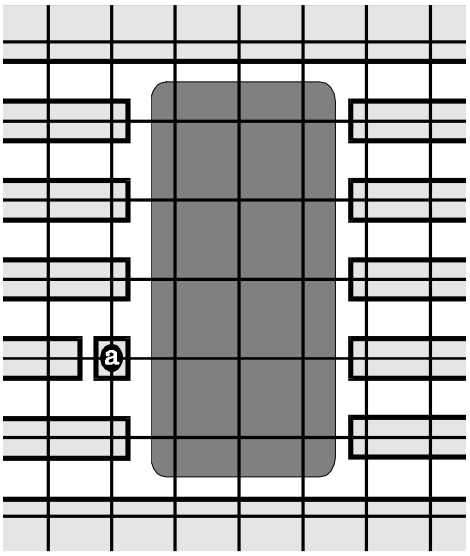


Figure 11: The hatching indicates (a cross-section of) the failed convex region, the light grey areas the necessary intervals at some node a. Node a needs $2(y_{ul}-y_{lr})$ extra intervals.

It is possible to route messages from any node on L, to any other node on R by using shortest paths. One then has to use $2(y_{ul}-y_{lr})$ intervals on each node (figure 11). Consider a node a of L. At node a, not only the messages for $R_a(\backslash CR)$ need to be redirected, but also messages for each interval $I \in \{I = R_u(\backslash CR) \mid u \in L_{za}\}$. For each I, the -Y or +Y direction has to be chosen to provide shortest path routing.

Analogous changes are of course necessary for the nodes of R.

These failures can also be handled by bypasses in the Z-direction. The necessary changes are completely analogous, but now messages remain in a horizontal plane of nodes with the same x-coordinate.

The number of extra intervals is proportional to the extent of the failed convex region in the Y-direction. An optimisation is obtained when a complete relabelling of the system is executed, whereby the X-, Y- and Z-directions are switched.

3.3.2. Cycle routing

First messages are routed to the correct Z-plane, in the same way as the above mentioned border routing. Next messages are routed in a cycle around the failed region, as indicated in figure 12. This means that only one interval is required to route the messages around the failed convex region. This technique can be used to overcome complex failures by using no extra intervals.

Compared to border routing, much less intervals are required. The routing is however less optimal.

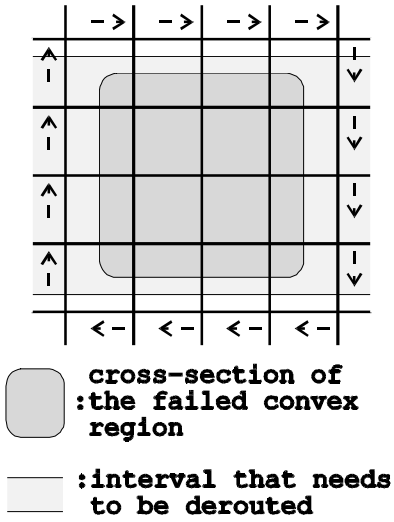
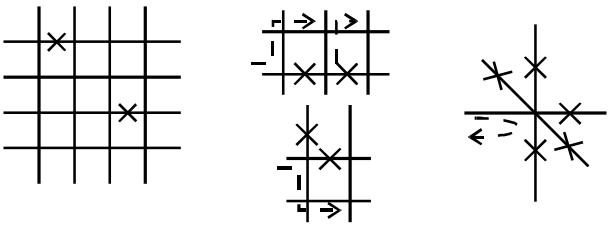


Figure 12: Cycle routing to overcome convex region failures.

3.4. Multiple link/node failures

We have three possible classes of multiple failures (figure 13):



not interfering fault | sequential fault | extension fault

Figure 13: Three types of multiple faults.

Not interfering faults are sufficiently separated to be handled separately. No special measures are required.
Sequential faults can, though they are neighbouring, be solved by handling them as consecutive single link faults. Attention should be paid to ensure that changes are made consistently with previous link faults and the newly created paths to overcome these failures.

Extension faults: These failures can not be handled by a single superposition of the reroutings for the individual link failures.

3.4.1. Superposition faults

Since not interfering link faults and sequential link faults can both be handled in the same way, we group these two classes together. We handle each individual single link failure sequentially, by applying one of the four possible local deroutings for this single link failure. Great attention must be paid to guarantee that this results in a global consistent routing scheme.

We solve this problem by sequentially recovering the individual link failures, taking care not to undo new paths created while recovering the previous failures. If one recovers a link failure, one imposes limitations for the nest recovery steps. These limitations must be taken into account to result in a consistent routing scheme.
As mentioned before, the splitting technique (section 3.1.4.) can be used to overcome inconsistencies. The main reason is that, by splitting intervals, one imposes less limitations. We illustrate this with the single node failure case (figure 5b). If we do not split interval D_b at node b into L_u and $D_b \backslash L_u$, we must ensure that the messages for L_u are routed around the node u to arrive well at their destination. This imposes lots of constraints, limitations to the recovery of the other link failures.

As we showed before, several alternative paths are available to recover from a link failure. This recovery consists of some jump over the failed link. As indicated for single link failures, some link failure l ($l = \pm X, \pm Y, \pm Z$) at node $\alpha = (x_{1\alpha}, ..., x_{i\alpha}, ..., x_{n\alpha})$ can be solved by rerouting the messages of the affected interval $I_{\alpha,l}$ to some node $\beta = (x_{1\beta}, ..., x_{l\beta}, ..., x_{n\beta})$, where $x_{l\beta} = x_{l\alpha} + sign(l)$ and $|x_{l\beta} - x_{l\alpha}| \leq 1$ (figure 14).

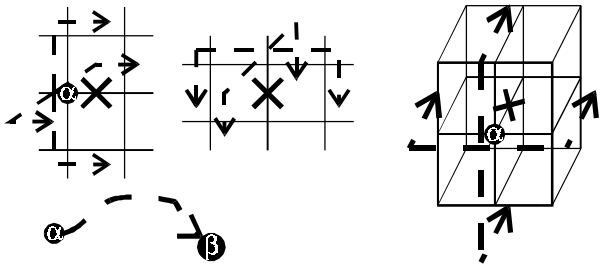


Figure 14: A link failure at node α can be handled by four alternative paths to a node β .

This approach can be justified in the following way. We start from a non-faulty system. The routing scheme is valid. Then a link l at a node α fails. Interval $I_{\alpha,l}$ can thus no longer be reached. We thus look for an alternative path to a node β and implement this new path. We then have a valid routing scheme again, since from β messages will correctly be routed to $I_{\alpha,l}$ and do not return to α, since $x_{l\beta} = x_{l\alpha+sign(l)}$. When a following link failure occurs, we apply the same strategy but take care that no inconsistencies arise, i.e. we do not undo the alternative paths made for previously recovered link faults. The previous link faults thus remain recovered, and we have a new valid routing scheme. This technique can be repeated until all faults are handled.

It is possible that we can not find such a node β (figure 15). Starting from node α, there is no possible path from node α to some node β. Then, the superposition of the adaptations for the individual link failures is not guaranteed to give a consistent solution.

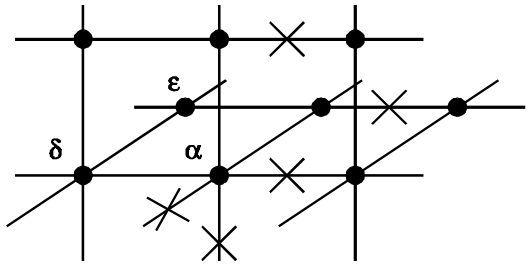


Figure 15: Superposition is not always possible.

In these cases, we decide that the faults are extension faults.

This results in the following algorithm :
Each node checks its own link failures. These link failures are then sequentially handled by using the basic rerouting techniques for the single link failures. Of course, previously changed paths must not be altered in order to guarantee consistency. If the rerouting requires adaptations on a neighbouring node, perform these changes too. If this simple superposition of rerouting techniques for the individual link failures does not work, decide upon an extension fault (section 3.4.2).

3.4.2. Extension faults

These failures can not be handled by a single superposition of the reroutings for the individual link failures. We thus construct a convex region around these failed links and nodes and route the messages around this convex region, just as for the failed convex region case.

Construction of the convex region:

- Each failed node belongs to a convex region (of possibly only one node)
- Each node without alternative paths to overcome its link failures is put into a convex region.
  All of its links are temporary considered to be faulty. This affects other nodes. So they will possibly be attached to a convex region too.

Some nodes inside this convex region are still working and accessible. Though their communication bandwidth is extremely reduced, so that they will slow down the whole system, we can still using the working nodes in the convex region. From the working node μ, we need to find a path to a node ν on the border of the region. This requires one interval on each node on that path. Each node α of the border, lying in the same plane as ν, needs to be adapted to reroute messages for μ to ν. This requires at most two extra intervals on α.

We illustrate the notion of extension faults with an example. If we have a closer look at node α of figure 14, we see that this node has only two remaining links left. To reroute messages, using a superposition of the simple alternative paths, we should be able to deroute the messages already at node δ. At δ we can send the messages to ε. This is a simple alternative path, causing no complications. In fact, at node δ we consider the whole node α to be faulty, though this is not the case. We decide upon a larger unit to be faulty.

4. Routing to spares

When a node fails or when its computational unit is no longer usable, we need to find an other node to execute the processes, previously running on the failed node. Therefore spare nodes are needed. Since all nodes may send messages to the failed node, and now thus to the spare, a system-wide adaptation is necessary.

Until now, we implicitly assumed that the application was running on a convex region. It is however possible that spares are only available at the border of this convex region. In that case, they have to be attached to the convex region at recovery-time. As a result, either a new convex region can be formed, or the spares are attached to the convex region as a bulge.

A first approach is to give the spare processors the same label as the failed processor they replace. Then the data on the other processors need not to change. At every node, we redirect the messages to their new position on a shortest path. To realise this at most two extra intervals are necessary on these nodes.
In a second approach, we let the spare processors have other labels than the processors they replace. It is then possible to route to the spares with no extra intervals: Suppose we need a partition of $A \times B \times C$ nodes. Now ψ nodes fail. We thus need ψ extra nodes. If we get a new convex partition, we can relabel the whole system of $A \times B \times C + \psi$ nodes as indicated in section 2.2.1. Then no special measures are needed to route messages to the spares. Of course the necessary changes must be done to reroute around the failed units, and the mappings of logical processors to physical ones need to be adapted.

Second, we consider the case where spares belong to the convex region from the start. This means that messages can already from the start be routed to the spares. The spares have a different label than the processors they will replace. At recovery time, only information in the processors, the mapping of logical processors to physical ones, has to change. Of course the allocation of spares results in a less efficient use of the system.

5. Conclusion

In this paper we presented several methods to overcome network failures in a 3D-grid communication network based on multi-interval routing. The resulting routing is near-optimal and requires only a limited number of extra intervals on a limited number of nodes. The number of extra intervals is independent of the network size. Reconfiguration is only necessary in a small area around the fault. This makes the approach very suited for scalable designs. It should be clear that the number of intervals in the routing device is a limiting factor for recovery.

A single link failure and a single node can be handled while using at most two extra intervals. Convex region failures can be handled by "border routing", requiring twice the number of nodes in the vertical dimension extra intervals, or by "cycle" routing, requiring no extra intervals. Multiple link failures can be handled by superposition, require at most two extra intervals per failure or can be handled by considering a larger system part to be faulty, using the convex region failure approach. We also presented routing schemes to access spares. On the nodes at most two intervals are needed.

We are currently working on a deadlock-free fault-tolerant routing scheme in grids, based on interval and wormhole routing.

References

[1] Jan van Leeuwen, R. B. Tan, *Routing with Compact Routing Tables*, Technical Report RUU-CS-83-16, Rijksuniversiteit Utrecht, November 1983

[2] W. J. Dally, *Wire-Efficient VLSI Multiprocessor Communication Networks*, Hingham, MA: Kluwer, 1987

[3] Jan Van Leeuwen, R. B. Tan, *Interval Routing*, The Computer Journal, vol. 30, no. 4, 1987, pp. 298-307

[4] N. Santoro, R. Khatib, *Labelling and Implicit Routing in Networks*, The Computer Journal, vol. 28, no. 1, 1985, pp. 5-8

[5] G. N. Frederickson, R. Janardan, Designing Networks with Compact Routing Tables, Algorithmica, vol. 3, 1988, pp. 171-190

[6] Inmos Ltd, The T9000 transputer, Products Overview, Manual, first edition 1991, pp. 139-162

[7] Technical Summary, Parsytec GC, version 1.0, 1991, Parsytec GmbH, Aachen, Germany

[8] G. N. Frederickson, R. Janardan, Space-Efficient and Fault-Tolerant Message Routing in Outerplanar Networks, IEEE Trans. on Computers, vol.37, no.12, 1988, pp. 1529-1540

[9] P. Kermani, L. Kleinrock, Virtual Cut-Through: A new Computer Communication Switching Technique, Computer Networks, vol. 3, pp. 267-286, 1979

[10] E. M. Bakker, J. van Leeuwen and R. B. Tan, *Linear interval routing schemes*, Technical Report RUU-CS-91-17, Rijksuniversiteit Utrecht, Februari 1991

[11] David May, Peter Thopmson, *Transputers and Routers: Components for Concurrent Machines*, Inmos Ltd, April 4, 1990