Implementation of Finite Lattices in VLSI for Fault-State Encoding in High-Speed Networks

Andreas C. Döring, Gunther Lustig

Medizinische Universität zu Lübeck
Institut für Technische Informatik
Ratzeburger Allee 160
23538 Lübeck, Germany
{doering,lustig}@iti.mu-luebeck.de

Abstract. In this paper the propagation of information about fault states and its implementation in high-speed networks is discussed. The algebraic concept of a lattice (partial ordered set with supremum and infimum) is used to describe the necessary operation. It turns out that popular algorithms can be handled this way. Using the properties of lattices efficient implementation options can be found.

1 Introduction

The constantly decreasing prices of computer hardware have made the building of ever larger computer systems more attractive. The interconnection between the components like storage media, memory, processing elements and graphics system has consequently migrated from busses to networks. Even inside a single computer the I/O-subsystem will consist of a high-speed network in the near future. The high bandwidth and low latency of modern network switches require an operating mode without or only with small software protocols. To achieve this, the network has to act very reliably. Hence, some of the actions to maintain operation in the presence of faults of some network components (routers or 'links', i.e. wires) have to be performed in the network itself. Toward this aim intensive research has been done. The task of allowing a network to circumnavigate faults consists mainly of two independent problems:

1. Detect the faults and propagate the knowledge about them through the network, and
2. use this information to select fault free routes for the messages.

In this paper the first problem is considered from a general point of view. The high variety of applications of networks makes a universal router desirable that can be configured for a given problem. In particular, both tasks mentioned have to be implemented in a configurable way. The investigation of hardware structures fulfilling this, leads to a description and implementation method for a wide range of routing algorithms [DOLM98].

Though failure of a network component is expected to be rare, the reaction of the network has to be very fast. This is because a new fault leaves the network for some time in an inconsistent state. This problem has been excluded in most algorithms by assuming a separation of the diagnosis phase (item 1 above) and the message transport. Many applications may not allow this procedure requiring additional methods like transporting affected messages to the nearest node and retransmitting them. The faster the fault state propagation in the network is, the fewer messages will have to be dealt with this way. Hence in this paper the necessary operation is considered with the aim of a hardware implementation.

With respect to the knowledge base needed to provide connectivity in a faulty network, fault-tolerant routing algorithms can be divided into local- and global-information-based. Global-information-based algorithms rest their routing decision upon fault-information that contains the location of any faulty component in the network. Hence optimal routing decisions can be made at intermediate nodes in the presence of faults. The problem of those algorithms is to ensure a consistent information base among all nodes of the network. This leads mostly to an off-line re-configuration of the whole system after failure detection.

The routing decision of local-information-based algorithms uses only knowledge about the router's own and the neighbors' fault states. This could result in a back-tracking for messages trapped in a dead-end [TW98, CS90]. In order to avoid back-tracking despite of the topological irregularities resulting from faulty components, some algorithms relay on a convex fault model. To achieve this with only local knowledge a router is allowed to pretend a fault although it is fault-free. Each connected set of faults is completed in this way to a convex shape (e.g. rectangular in 2D meshes/tori or cubic in 3D meshes/tori). This transforms the remaining topology to a more regular one by adding fault states over a certain distance and enables messages to bypass the faulty regions on detour routes.

Alternative approaches e.g. [Wu98, CW96, CA95] use limited global information in their routing decision. Beside the local node state, the routing decision uses information from neighbor nodes that allows conclusion about the reachability of even non-adjacent regions in the network. To generate this knowledge, each router calculate its own state dependent on local faults and its neighbors' states. Clearly, it takes some time until all nodes have adjusted their own state to a new fault and several state exchanges and updates are needed.

The larger the set of possible states a router can take is, the better the knowledge about the distribution of faults in the whole network can be. Better knowledge allows tolerating more faults, keeping the performance in presence of faults higher or reducing the number of nodes that cannot be reached by the routing algorithm (like those in the convex fault model that have to be marked faulty). A larger state set requires more hardware (memory) to store it, bandwidth to exchange it, and processing power to handle it. The processing of state information is the topic of this paper.

It appears that no general method or framework dedicated for the fault state propagation has been developed, because routers implemented so far are either done entirely in software, or use no or only one fault tolerance method[AGSY94]. The flexible implementation of the fault state propagation requires special methods due to the strong speed requirements.

A large number of routing algorithms working with a limited global knowledge reveals the observation that the set of states can be described by the mathematical concept called *Lattice*. Especially the update of a node's state is essentially the computation of the supremum operation in the lattice. This concept is described in section 2. To illustrate the approach, the fault state propagation of two algorithms is used in section 3. Some proposals for a hardware implementation are given in 4 followed by the conclusion (section 5).

2 Lattices and Fault-Tolerance

The notion of a lattice is well known in algebra for a long time. For different purposes advanced theories have been created. There are two meanings for the term lattice, namely a discrete subset of a vector space (a grid) and a set with a partial ordering. In this paper the second meaning is understood.

Fault-tolerant systems are usually modeled as construction of components which are all or partially vulnerable to defects (faults). Hence, the state with respect to the faults is described by a combination of fault states from the individual components. The individual fault states have different implications to the functionality of the whole system. With respect to this influence and the methods applied in which the system reacts to the state a partial order can be defined. If some situation is strictly "worse" than another, it has a higher order of "defectness". More precisely, if all consequences for mal-function or poorer performance from the first situation apply to the second and if at least the same actions (repair etc.) have to be taken, than both situations can be compared. Of course there are situations where two different properties of the whole system are affected and where the necessary actions also vary. This notion of "worse" justifies the application of the concept of lattices to the handling of fault states.

Since a certain fault state of a certain router is less critical for its neighbors, the transmitted value has to be converted. To incorporate geometric information all neighbors get different information, see for instance figure 3 in section 3. The state of total fault of a node is transformed into North-,South-, East- or West-Failure respectively. These mappings are lattice-homomorphisms. Since they are less complex than the generation of the new fault state they are not further considered in this paper.

Definition 1. *A lattice is a tuple* $(M, \vee, \wedge)$ *of a set* M *and tow functions* $\vee, \wedge : M \times M \mapsto M$ *which fulfill*

$$\forall m, n \in M : \wedge(m,n) = \wedge(n,m), \vee(m,n) = \vee(n,m)$$

$$\forall l, m, n \in M : \wedge(l, \wedge(m,n)) = \wedge(\wedge(l,m),n), \vee(l, \vee(m,n)) = \vee(\vee(l,m),n)$$

$$\forall m, n \in M : \wedge(m, \vee(m,n)) = m, \vee(m, \wedge(m,n)) = m$$

The two functions are also called supremum sup *and infimum* inf *.*

The intuitive meaning is that the elements of a set M are partially ordered, where all pairs of elements (m,n) have a unique smallest upper $\vee(m,n)$ and lower $\wedge(m,n)$ bound. In the following only lattices with a finite set M are considered. This is motivated by a system model with a finite number of points of failure. In every finite lattice a unique smallest $\perp$ and a unique largest element $\top$ exists. Furthermore the supremum $\bigvee_S$ and infimum $\bigwedge_S$ is well defined for arbitrary subsets S of M. A chain in $\mathfrak{L}$ is a sequence of distinct elements $m_1, \ldots, m_k$ where $\vee(m_i, m_{i+1}) = m_{i+1}$. It is usual to illustrate partially ordered sets as a Hasse-diagram which is a directed Graph (M, E). The edges E in this graph are given by directly comparable relations: $(m,n) \in V \Leftrightarrow \wedge(m,n) = n$ and $\forall l \in M : (\wedge(m,l) = l$ and $\wedge(l,n) = n) \Rightarrow l = n$

Edges of the Hasse-diagram can be viewed as generators of the lattice. The nodes can be labeled ($\alpha : M \to \mathbb{N}$) with the distance to to the top ($\top$) which results in a layered representation. From $\wedge(m,n) = m$ follows that m is in a higher level than n. More generally, $\alpha(\wedge(m,n) \geq \max(\alpha(m), \alpha(n))$.

For the application to fault-tolerance a special class of lattices is of interest, lowest-level generated lattices:

Definition 2. *A lattice* $(M, \vee, \wedge)$ *with Hasse-diagram* (M, E) *is called lowest-level generated (llg) iff* $M = \{\perp\} \cup G \cup \{\bigvee_S | S \subset G\}$
where $G = \{g \in M | (\perp, g) \in E\}$

From an algebraic point of view this means that the set of points which are immediately larger than the bottom element generate the whole lattice already by the $\wedge$, of course except the bottom element. The importance of this class is that it exactly reflects the situation of fault-states sketched before. The single faults of components in the system represent the generating set G. The only better situation than two different single faults is a system in order. Furthermore if the set of faults are the only parameter that induces the fault state of the system all elements of the lattice have to be generated by the set G.

There is a simply criterion whether a lattice is llg or not.

Lemma 1. *A lattice* $\mathfrak{L} = (M, \vee, \wedge)$ *with Hasse-diagram* (M, E) *is llg iff the in-degree (number of edges to a node) of all elements from* M *except* G *and* $\perp$ *is greater than one.*

$$\forall x \in (M \backslash G) \backslash \{\perp\} : |\{(y, x) \in E\} > 1$$

Some important examples are given now, where all but the first one are generally llg:

1. $\mathfrak{B}(n) := (\{\text{True}, \text{False}\}^n, |, \&)$ Here & denotes the conjunction and | the disjunction, i.e. digit-wise AND respectively OR. The top element is the vector with all True and the bottom element with all False. Clearly, this lattice is llg, since the lowest level consists of all vectors with just one place True. It has 2^n elements.
2. $\mathfrak{N}(n) := (\{0, \dots, n \Leftrightarrow 1\}, \max, \min)$ Every finite linearly ordered set with n elements is isomorphic to this lattice. If $n > 2$ it is not llg, since the generating set consists of only a single element.
3. $\mathfrak{M}_k(n) = (\{T \subseteq \{1, \dots, n\}, |T| \leq k\} \cup \{\{1, \dots, n\}, \cap, \cup_k)$
   where $S_1 \cup_k S_2 := S_1 \cup S_2$ if $|S_1 \cup S_2| \leq, \{1, \dots, n\}$ otherwise
   This the the frequently-found max-k-faults failure model. More than k faults make the whole system faulty.

Though there has been intensive research on lattices the only information about the occurrence frequency seems to be found in [Kyu79]. In this paper the number of different lattices with up to 9 elements is elaborated by algorithmic enumeration and isomorphy checking. We re-implemented this algorithm in order to get further information. Surprisingly some more graphs have been found. The results are given in the following table. It appears that the number of llg lattices is only a small fraction of the lattices of a given size.

$\|M\|$	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
#lattices	1	1	1	2	5	15	53	223	1100	6330	42155	$\cdots$
# llg	1	1	0	1	1	2	4	9	22	60	193	$\cdots$

Up to $l = 4$ elements in the lattice there are only the Boolean lattices which are llg. Hence, there is no llg lattice with three elements. Though the table does not suggest it the number of llg lattices may also grow strongly. Any lattice can be made llg with adding some elements. The resulting lattices have too many elements to be found in the table. How many of them are non-isomorphic remains open.

3 Application to Selected Fault-Tolerant Routing Algorithms

By exchanging status information with its neighbors, a router node determine whether it is part of a faulty region. Figure 3 illustrates the lattice for routing algorithms which allow the overlapping of detour routes on the border of fault regions. The router is a neighbor node of a fault region if it detects only one of his neighbors to be faulty or in the state $\top$. A combination of two faults in the same dimension leads to a state that indicates the router to be adjacent to distinct fault regions. The case that two or more failures are detected in different dimensions results in the supremum. This means that the node itself is part of a fault region and reaches the state $\top$.

The approach of convex fault regions offers a simple solution for low dimensional topologies. In higher dimensional topologies the link redundancy enables

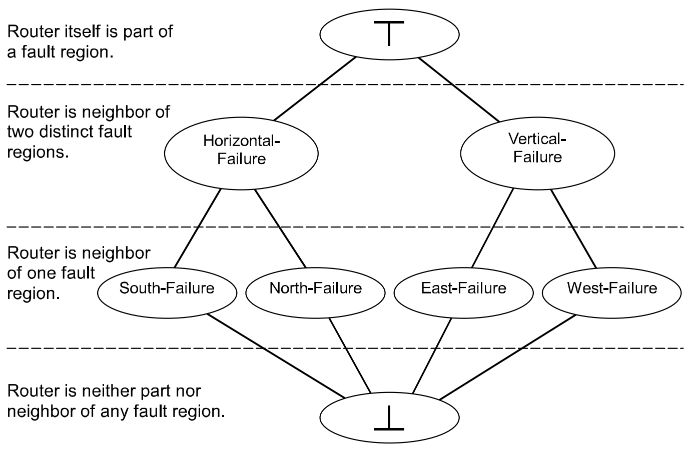
Router itself is part of a fault region.
⊤
Router is neighbor of two distinct fault regions.
Horizontal-Failure
Vertical-Failure
Router is neighbor of one fault region.
South-Failure
North-Failure
East-Failure
West-Failure
Router is neither part nor neighbor of any fault region.
⊥

Fig. 1. Fault-lattice for convex regions in a 2D-mesh.

fault-tolerance concepts that allow the avoidance of detours in advance. As an example the formation of a fault lattice in hypercubes to the concept of [Wu98] is used. Within this approach the fault information is captured in a safety vector of n bits. A node where the k-th bit of the safety vector is set, guarantees message transportation on a minimal path for all destinations with Hamming distance k. To calculate the vector within each node, an information exchange of $n \Leftrightarrow 1$ rounds is necessary. Based on a topological property of the binary hypercube, the k-th bit of the safety vector (signed with S in the following) is determined from the $(k \Leftrightarrow 1)$th bit of the neighbors' safety vectors. The first bit of each safety vector is initialized with respect to the own node state.

Figurge 3 shows the resulting lattice for determining the second bit of a safety vector in a fault free node. In order to concentrate on the concept, a hypercube with only three dimensions has been chosen.

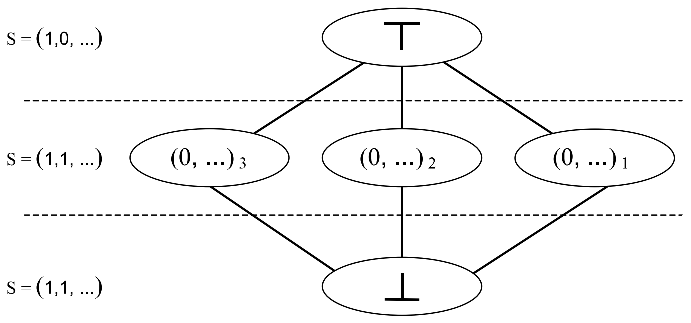

Fig. 2. Fault-lattice for determination of safety vectors 2nd Bit.

For calculating the second bit of the safety vector only the first bit of the neighbors' vectors $(0, \ldots)$ or $(1, \ldots)$ are essential. The safety vector of the i-th neighbor is indicated with the index i. As neighbors' safety vectors with the first bit is set to one do not change the status of the own safety vector, these transitions are dispensable for the fault-lattice. After receiving one safety vector containing a zero at its first bit, the resulting vector does not differ from its initial state. Only the receipt of two or three safety vectors where the first bit was set to zero results in an unsetting of the second bit in the own safety vector.

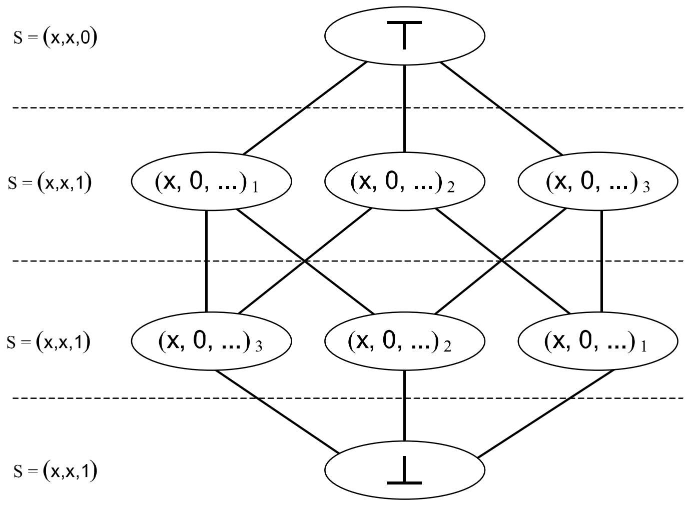

Fig. 3. Fault-lattice for determination of a safety vector's 3rd Bit.

Calculation of the third safety bit in Figure 3 is similar to the second one, except that the result has only a zero if all adjacent nodes have set their second bit to zero. Many other routing algorithms can be treated this way but as the resulting lattices are larger an abstract approach to their description is needed, allowing an automated processing. The rule-based approach presented in [DOLM98] serves this purpose.

4 Implementation

In this section the configurable implementation of finite lattices $\mathfrak{L} = (L, \vee, \wedge)$ in hardware is considered. The description of the implementation is scalable with respect to the size of the lattice $l = |L|$. Furthermore only the implementation of one function, say $\vee L \times L \rightarrow L$ is discussed, because this is sufficient for fault-tolerant routing algorithms. All proposed implementations can be easily extended to also include $\wedge$. Another assumption is that the encoding of the elements of L into a bit vector can be freely chosen.

Only few results seem to be known about the complexity of implementing the supremum in lattices. For the two trivial cases – the Boolean lattice and sets of integers – the implementation is straight-forward: n-bit OR and MAX respectively. Both need $O(\log l)$ gates and $O(1)$ respectively $O(\log\log l)$ depth. Layout questions are widely discussed since both are frequently used functions. However both circuits are quite different and there is no obvious configurable super-circuit which could implement both problems.

4.1 Simple Table-Based Method

A reference design is given by a look up table with l^2 entries. It has an area of $O(l^2 \log l)$ and a depth of $O(\log l)$. Reflexivity ($\vee(x, x) = x$) and commutativity

($\vee(x,y) = \vee(y,x)$) can be used to reduce the size of the table t. For the latter, any order $<$ can be used to "reflect on the matrix diagonal":

$$\vee(x,y) = \begin{cases} x & \text{if } x = y \\ t(x,y) & \text{if } x > y \text{ and } l/2 > y \\ t(l \Leftrightarrow x, l \Leftrightarrow y) & \text{if } x > y \text{ and } l/2 > y \\ t(y,x) & \text{if } y > x \text{ and } l/2 > x \\ t(l \Leftrightarrow y, l \Leftrightarrow x) & \text{otherwise} \end{cases}$$

The indexing of the table t assumes that the encoding of the elements uses a continuous sequence of binary encoded integers. In this way, the table t has dimensions $0, \ldots, l \Leftrightarrow 1$ for the first, and $0, \ldots, \lceil((l \Leftrightarrow 1)/2)\rceil$ for the second index. How these two indices can be combined into a single address for a memory in a chip is a more general topic beyond the scope of this paper. The resulting circuit is dominated by the $l(l \Leftrightarrow 1)/2\lceil\log l\rceil$ memory cells. In a limited fan-in model the depth of this circuit is $O(\log\log l)$ for the comparisons and the memory addressing. Since the order relation of $\mathfrak{L}$ can be presented as an adjancy matrix, $l(l \Leftrightarrow 1)/2$ memory bits would be sufficient. However the computation of the $\vee$ function is rather difficult in such a presentation.

Since it seems to be not yet sure that the asymptotic behavior of the number of lattices for a given l is considerably smaller than $O(c^{l^2})$ this might be already an asymptotically optimal implementation up to constant factors. Hence, in the following only implementations which do not necessarily cover all lattices for a given l are considered.

4.2 Implementation with Boolean Lattice

The Boolean lattice is universal, which means that every lattice can be presented as a sub-lattice of the boolean lattice. However this can be quite inefficient: the lattice $\mathfrak{N}$ (n) with n elements would have to be implemented by the Boolean lattice with 2^{n-1} elements. Hence, a reduction is desirable for an implementation on base of $\mathfrak{B}$ (l-1). For most lattices there already exists an inclusion into a smaller Boolean lattice $\mathfrak{B}$ (m). The circuit consists of three parts:

1. Two encoders which generate an injective mapping $\iota : L \to \mathfrak{B}(m)$ from L into the Boolean lattice of m bits(configurable).
2. The circuit for $\vee$ in the Boolean lattice (m OR-gates).
3. A decoder $\mu : \mathfrak{B}(m) \to L$ which generates again elements in L (also configurable).

Since the correspondence of the top- or bottom element is given, it is clear that $m = l \Leftrightarrow 1$ if all possible lattices with l elements shall be implementable. For doing this, the digits of the vectors in $\mathfrak{B}(l \Leftrightarrow 1)$ are indexed with the elements of $\mathfrak{L}$ except $\top$. The image $\iota(x)$ of an element x of $\mathfrak{L}$ is the vector where the digit d_y is True iff $x \vee y = y$, i.e. for all elements greater or equal than x in $\mathfrak{L}$. Hence conjunction results in an element of $\iota(m)$.

Encoding of the elements of M can be chosen in a way that minimizes the size of the encoder circuit. Since the codes for the top and bottom elements are fixed there are $l \Leftrightarrow 2$ codes to be chosen. One further element can be fixed: there has to be at least one element s in M which is immediately larger than bottom, i.e. no other element but bottom is smaller than s. Hence, this element can be mapped to the vector $(1, 0, \ldots, 0)$ in $\mathfrak{B}(n)$. Looking at the both lattices $\mathfrak{N}(n)$ and $\mathfrak{M}_2(n \Leftrightarrow 2)$ it is obvious that there is no further similar simplification, since in the first case all other elements are comparable to each other while in the second example no two of the remaining elements are comparable.

The implementation of μ uses a standard circuit namely a priority encoder. This is possible because every element of $M\backslash\{\top\}$ refers to a digit in $\mathfrak{B}(l \Leftrightarrow 1)$. The order of M induces an order on the digits. Finding the right element in $\iota(M)$ is identical with finding the lowest digit in the resulting vector which is True. In consequence the decoder (consisting of an priority encoder) does not need to be configurable. Since every partial order can be embedded in a total order this is fairly possible.

Because the bottom element has no corresponding digit, the highest input of the priority encoder (of l bits) is fed with constant True, reflecting that $\top$ is larger than any element of M. A similar simplification can be done for $\perp$ if $l > 2$. In this case the test for the result $\vee(\perp, \perp)$ saves another bit per vector. It remains to discuss options for the implementation of ι. If two tables are used (with diagonal reflection), the resulting circuit needs $(l \Leftrightarrow 2)(l \Leftrightarrow 3)/2$ memory bits. The memory has to be dual ported, that is the multiplexers for output selection are needed twice.

However, for many applications including fault-tolerant routing algorithms a much smaller m can be sufficient because typical applications have shorter chains. Especially llg lattices can be included in $\mathfrak{B}$ (g), where g is the number of generating elements.

The interesting question is the implementation of the encoder and decoder circuit. For instance the top and bottom elements of the lattice can be assigned with fixed codes removing any need for configuration. Though the encoder is found twice in the design, the problematic part is the decoder because it has to map from the much larger $\mathfrak{B}(n)$ onto L. The mapping has to be consistent with the order of L, i.e. it has to be an partial-order endomorphism. In most cases $C = \vee(\iota(a), \iota(b)$ will be no element of $\iota(M)$ for arbitrary $a, b \in M$. More precisely, if for all pairs a, b C is in M then ι is an lattice endomorphism which is an unnecessary strong restrictions for computing $\vee$ alone.

Hence, in the more general case the implementation of $\mu(C)$ has to find the smallest element of $\iota(M)$ which is larger than C with respect to the order in $\mathfrak{B}(m)$. For an efficient implementation ι has to be chosen in a way which makes this step easy. Fortunately there are only the two restrictions of monotony and injectivity on ι.

4.3 Hybrid Implementations

Since lattices are a category of universal algebras, constructs like direct products are available. These constructions carry over to the implementation: direct products of lattices can be computed by the independent calculation of its factors. A direct product represents the combined state of two independent systems. A more interesting option is the identification of sub-lattices, because this occurs much more frequently. In this context two different ways of embedding a sub-lattice are distinguished. A sub-lattice $\mathfrak{A} = (A, \vee, \wedge)$ is said to be included point-like in a lattice $\mathfrak{L}$ if the only edges in the Hasse-diagram to nodes outside the sub-lattice are adjacent to $\mathfrak{A}$'s top and bottom elements. Formally

$$\forall x \in M\backslash A, \forall a \in A : \vee(a, x) = \vee(\top_{\mathfrak{A}}, x) \text{ and } \wedge(a, x) = \wedge(\perp_{\mathfrak{A}}, x)$$

In Figure 4 two sub-lattices are shaded where only one (denoted “Lattice B”) is embedded point-like.

At least small point-like sub-lattices can be found in every lattice. Especially with respect to fault-tolerance it can be expected that a lot of typical point-like included sub-lattices can be found. This is interesting from an implementation point of view because optimized subcircuits for the special sub-lattice can be used. To allow more lattices to exploit these fixed implementations by adding only a small amount of hardware some selected points with edges out off the sub-lattice can be permitted.

For a configurable implementation for large lattices with a suffient fraction of typical sub-lattices an implementation is suggesting which consists of two parts:

1. A pool of circuits for the implementation of typical sub-lattices, e.g. $\mathfrak{M}_k(n)$, $\mathfrak{N}(n)$, etc. These components can be configured only in a very restricted way, e.g. the constants n,k and the selection of some special elements.
2. An implementation of the “super-lattice” by a universal methods like those described before.

A configurated implementation of the example lattice can be seen at the right in figure 4. It is interesting to observe that the resulting architecture resembles strongly the implementation approach for the whole routing algorithm given in [DOL98].

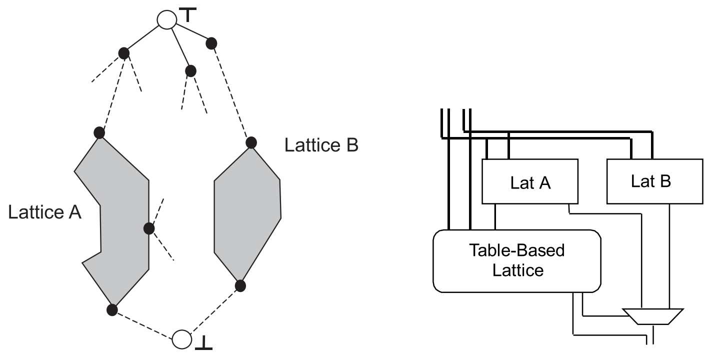

Fig. 4. A composed lattice and its implementation

5 Conclusion

In this paper it has been shown that the fault state propagation of many routing algorithms can be described by a lattice. A special class of lattices has been identified which is especially important for this application. In a discussion of implementation methods some VLSI architectures have been proposed. Of course, this method can also be used in a software implementation. In the latter context it is interesting to look for special machine instructions for a network processor to support these kind of problems.

Since most routing algorithms do not provide complexity measures for the state propagation, the overhead implied by the interpretation as a lattice can not be answered directly. Conversely, properties of the lattice (largest point-like sub-lattice, number of chains, level sizes etc.) can be used to judge various approaches.

The main disadvantage of the approach in comparison to an ad-hoc solution is that the sequence of arriving fault information can only hardly be exploited. On the other hand this gives robustness when some of the exchange messages get lost. Furthermore different routing algorithms may profit from the placement of the ‘reducing homomorphism’ which transforms the state of one node into an appropriate value for the neighbor’s state calculation. Further questions remaining open are:

– asymptotic frequency of lattices and lowest-level generated lattices
– For networks with some hundreds of nodes routing algorithms with global knowledge can be still practical. Does the better behavior justify the higher demand on bandwidth, memory and processing effort in contrast to limited-global routing algorithms. Furthermore routing algorithms with global knowledge tend to do a lot of redundant calculations.

– For many topologies only a few schemes of reduced fault state representation are known. More detailed methods like the one in NAFTA ([CA95]) could not be found in the literature for many important topologies, like the star graph.

The common framework of lattices can be integrated into tools for the evaluation simulation and optimization of routing algorithms and routers. This is simplified by using results, algorithms and tools from algebra.

In the project RuBIN ("Rule Based Intelligent Networks") the authors have implemented a translation tool for a high-level description method onto dedicated hardware structures. These hardware structures are implemented in a FPGA-based prototype using 1 million gates of programmable logic [Xil98] and myrinet link technology [BCF+95].

References

[AGSY94] James D. Allen, Patrick T. Gaughan, David E. Schimmel, and Sudhakar Yalamanchili. Ariadne — an adaptive router for fault-tolerant multicomputers. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 278–288, Chicago, Illinois, April 18–21, 1994. IEEE Computer Society TCCA and ACM SIGARCH.

[BCF+95] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1), Februrary 1995.

[CA95] Chris M. Cunningham and Dimiter R. Avresky. Fault-Tolerant Adaptive Routing for Two-Dimensional Meshes. In *Proceedings of the First International Symposium on High-Performance Computer Architecture*, pages 122–131, Raleigh, North Carolina, January 1995. IEEE Computer Society.

[CS90] Ming-Syan Chen and Kang G. Shin. Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):152–159, April 1990.

[CW96] Ge-Ming Chiu and Shui-Pan Wu. A Fault-Tolerant Routing Strategy in Hypercube Multicomputers. *IEEE Transactions on Computers*, 45(2):143–155, February 1996.

[DOL98] A. C. Döring, W. Obelöer, and G. Lustig. Programming and Implementation of Reconfigurable Routers. In *Proc. Field Programmable Logic and Applications. 8th International Workshop, FPL '98*, volume 1482 of *Lecture Notes in Computer Science*, pages 500–504, 1998.

[DOLM98] Andreas C. Döring, Wolfgang Obelöer, Gunther Lustig, and Erik Maehle. A Flexible Approach for a Fault-Tolerant Router. In *Proc. Parallel and Distributed Processing - Proceedings of 10 IPPS/SPDP '98 Workshops*, volume 1388 of *Lecture Notes in Computer Science*, pages 693–713, 1998.

[Kyu79] Shoji Kyuno. An Inductive Algorithm to Construct Finite Lattices. *Mathematics of Computation*, 33(145):409–421, January 1979.

[TW98] Ming-Jer Tsai and Sheng-De Wang. A fully adaptive routing algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):163–174, February 1998.

[Wu98] Jie Wu. Adaptive Fault-Tolerant Routing in Cube-Based Multicomputers Using Safety Vectors. *IEEE Transactions on Parallel and Distributed Systems*, 9(4):321–334, April 1998.

[Xil98] Xilinx, Inc. XC4000XV Family Field Programmable Gate Arrays. Data Sheet, May 1998.