TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

A decision procedure for deadlock-free routing in wormhole networks

Citation for published version (APA):
Verbeek, F., & Schmaltz, J. (2014). A decision procedure for deadlock-free routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, *25*(8), 1935-1944. https://doi.org/10.1109/TPDS.2013.121

DOI:
10.1109/TPDS.2013.121

Document status and date:
Published: 01/01/2014

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 24. Jun. 2021

A Decision Procedure for Deadlock-Free Routing in Wormhole Networks

Freek Verbeek and Julien Schmaltz

Abstract—Deadlock freedom is a key challenge in the design of communication networks. Wormhole switching is a popular switching technique, which is also prone to deadlocks. Deadlock analysis of routing functions is a manual and complex task. We propose an algorithm that automatically proves routing functions deadlock-free or outputs a minimal counter-example explaining the source of the deadlock. Our algorithm is the first to automatically check a necessary and sufficient condition for deadlock-free routing. We illustrate its efficiency in a complex adaptive routing function for torus topologies. Results are encouraging. Deciding deadlock freedom is co-NP-Complete for wormhole networks. Nevertheless, our tool proves a 13×13 torus deadlock-free within seconds. Finding minimal deadlocks is more difficult. Our tool needs four minutes to find a minimal deadlock in a 11×11 torus while it needs nine hours for a 12×12 network.

Index Terms—Communication networks, deadlocks, routing protocols, automatic verification, formal methods

1 Introduction

Wormhole switching [1] is a popular switching technique where packets are decomposed in smaller units, called *flits*. The header flit contains routing information. The remaining flits consist of the payload and a tail flit indicating the end of the packet. Only the header flit is used for routing. The data and tail flits follow the header flit in a pipelined fashion. Wormhole switching provides low message latency and requires small buffer capacity in the channels of the network as buffers need not store whole packets. This switching technique is also prone to deadlock. The flits of a single packet often hold many resources simultaneously, blocking many other messages. Deadlocks are a key issue in the design of wormhole networks [2], [3].

A practical solution to guarantee deadlock freedom is to prevent deadlocks in the design of the routing function. The development of theories and methodologies for deadlock-free routing in wormhole networks has been a fruitful research area for many years [1], [4], [2], [5]. A typical example is the work of Duato for adaptive routing functions [2], [3]. Duato defined a necessary and sufficient condition guaranteeing the absence of deadlock. The methodology associated with this condition starts from a deadlock-free routing function and allows extensions provided that messages eventually are routed back to and stay in the deadlock-free subnetwork.

Duato's methodology has been very popular and supported the design of very efficient routing protocols. But, it is a manual process. It is error-prone and not scalable. Moreover, the methodology requires an initial deadlock-free network. Such an initial network is not always available, as demonstrated in the design of the *clue* routing algorithm [6]. In this case, a complex manual proof of several pages was needed to demonstrate the absence of deadlocks.

Our aim is to provide a *fully automatic* approach for deadlock verification in wormhole networks. Such an approach should have the following characteristics:

- Applicability: the method should be applicable to *any* reasonable topology and routing logic.
- Ease of use: the method should be push-button and users should provide little input.
- Scalability: results should be obtained quickly even for large networks. This really is a challenge as deciding deadlock freedom in wormhole networks is a co-NP-complete problem [5].
- Analysis: the approach should provide readable and understandable feedback explaining the cause of deadlocks.
- Sanity checks: common sanity checks must be performed on the input to prevent minor errors.

The contribution of this paper is the first necessary and sufficient algorithm for deadlock verification in wormhole networks. Our algorithm only requires as input a description of the routing function. The only assumption on the routing logic is that it is memoryless, i.e., it decides the routes based on the current position of the message and the destination. Our decision procedure decides whether this routing function satisfies Verbeek and Schmaltz' necessary and sufficient condition [5]. It returns, whenever the network is not deadlock-free, an exact description of the deadlock. This entails which messages participate, which channels they occupy, and for which destination they are headed. To ensure that this feedback is readable, we return the smallest deadlocks that can occur. Due to various optimizations, our approach scales to sufficiently large networks.

- *The authors are with the School of Computer Science, Open University of The Netherlands. E-mail: {freek.verbeek, julien.schmaltz}@ou.nl.*

Manuscript received 19 Nov. 2012; revised 31 Mar. 2013; accepted 13 Apr. 2013. Date of publication 18 July 2013; date of current version 16 July 2014. Recommended for acceptance by J. Flich.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.121

1045-9219 © 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

We incorporate our decision procedure in the tool DCI2 (for Deadlock Checker In Designs of Communication Interconnects).[1] DCI2 takes as input a small description of the topology and a C++ implementation of the routing logic. It checks for several minor errors such as a disconnected routing function or topology violations. It decides livelock freedom and uses the algorithm presented in this paper to decide absence of deadlocks. The result is sound and complete.

Our decision procedure consists of two steps. The first step, previously published [7], is a polynomial procedure which can quickly decide for many of the channels in the network whether they can participate in a deadlock. If Step 1 identifies a network as deadlock-free, Step 2 is not needed. Networks which, e.g., have no circular dependencies are proven deadlock-free quickly by this first step. Step 1 only checks a sufficient condition and may output false deadlocks. Typical false deadlocks consist of intersecting worms. An essential aspect of wormhole switching is that flits belonging to different messages cannot share channels. A deadlock must only involve a set of pairwise disjoint worms. Step 2 performs this analysis by reducing deadlock freedom to an Integer Linear Programming (ILP) instance.

Wormhole networks possess many subtleties making their analysis difficult. To ensure the correctness of our algorithm, it has been mechanically proven correct using the ACL2 theorem proving system [8]. In this paper, we will clearly point out which parts of the algorithm have been formally proven correct. Details about its formal verification can be found in a previous publication [9].

2 Related Work

A lot of research has been devoted to a search for *generic* theorems ensuring that a routing function is deadlock-free [1], [10], [2], [11], [12], [13]. Typically, the dependencies between channels are captured by a *dependency graph*. Early work by Dally and Seitz has shown that an acyclic dependency graph is a necessary and sufficient condition for deadlock-free routing [1]. This early theorem only applies to deterministic routing functions.

An interconnection network can have an adaptive routing function. If a message is blocked on its way, an adaptive routing function may supply an alternative next hop allowing further progress. Duato was the first to propose theorems for deadlock-free routing in adaptive networks [2], [14], [3]. He noticed that alternative paths could be used to *escape* deadlock situations and that a cyclic dependency is not a sufficient condition to create a deadlock [10].

Duato's theorem holds for a large class of wormhole networks. A significant part of the value of Duato's theorem is the corresponding design methodology [2], [15]. Consider a network and a routing function that are already known to be deadlock-free. New channels and routing capabilities can be added to this network as long as once a message arrives in an original channel, it cannot be routed towards new channels. Duato's condition holds for any network obtained in this way. The set of networks that can be proven deadlock-free using Duato's theorem is much larger than the set of networks that can be obtained using his design methodology. However, using Duato's theorem to show absence of deadlock of a network where his design methodology does not apply can require a highly complicated manual proof.

There are other design methodologies that ensure deadlock-free routing. Various design methodologies are based on a *turn model* [4], [16]. This model restricts the routing function in such a way that no cyclic dependencies occur. It was initially defined for two-dimensional (2D)-mesh topologies. Starobinksi *et al.* use the network calculus to generalize this methodology to general topologies [17]. Palesi *et al.* incorporate message dependencies, induced by a model of the applications running on the cores, into the dependency graph and provide a design methodology breaking all cycles [18].

These methodologies target static and fault-free networks. Duato, Lysne, *et al.* present a theory and a corresponding methodology to dynamically reconfigure a routing function in such a way that deadlock freedom is ensured [19], [20]. Their methodology applies to a broad class of interconnection networks, both on-chip and off-chip.

An inherent disadvantage of all these design methodologies is that they limit the designer. All these methods cannot be applied to arbitrary topologies and routing functions. Also, these methodologies are restrictive, in the sense that many deadlock-free designs are excluded. Most of these methodologies break all dependency cycles. As for adaptive routing a dependency cycle is not necessarily a deadlock, this is very limiting. Finally, these methodologies are manual and therefore error-prone.

As example, the next section introduces the *clue* routing logic introduced by Luo and Xiang [6]. Their routing function was not based on any of these design methodologies. Their only option to establish absence of deadlock of their design was by a manual proof, which took considerable effort.

In contrast, our approach offers a decision procedure for deadlock freedom of wormhole networks. Our approach applies to any topology and routing function, it is necessary and sufficient, it is fully automatic and formally proven correct.

To the best of our knowledge, the only other algorithm that detects deadlocks has been created by Taktak *et al.* [13]. The algorithm of Taktak *et al.* does not check a necessary and sufficient condition and is therefore subject to false deadlocks. The condition checked by Taktak *et al.* is logically equivalent to the condition checked by the first step of our algorithm. We have shown [7], [21] that our algorithm scales significantly better than the work of Taktak *et al.*

3 Motivating Example

We present the *clue* routing algorithm for torus networks, proposed by Luo and Xiang [6]. This routing function will be a running example throughout this paper. Its main characteristics relevant to this paper are the following:

- Clue allows circular dependencies. Any routing function without circular dependencies is easily and quickly proven deadlock-free by the first step of our

1. Available at: http://www.cs.ru.nl/~freekver/DCI2/

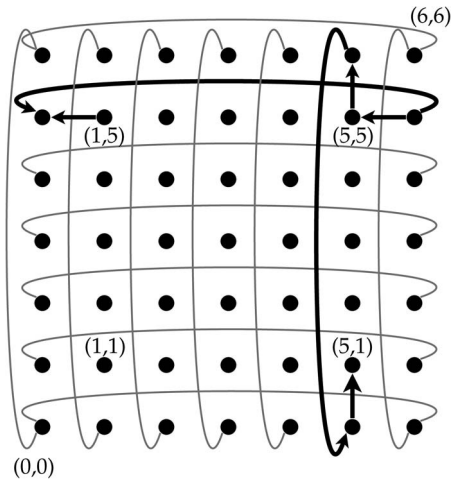


Fig. 1. Possible route supplied by clue logic.

algorithm, without using the analysis presented in this paper.

- Clue is deadlock-free in *packet* networks, but causes deadlocks in *wormhole* networks. If a routing function causes a deadlock in a packet-switched network, it is not necessary to use the analysis presented in this paper to determine deadlock freedom of the routing function in wormhole networks.
- Clue was not based on Duato's design methodology for routing functions [2]. Using Duato's condition to prove absence of deadlocks was not a practical approach. Instead, deadlock freedom has been established by a complicated, large and informal pencil-and-paper proof.

The topology targetted by the clue designers was an n-dimensional torus. For sake of presentation, we will focus on a 2D torus. Each processing node has four outgoing channels in the eastern, western, northern and southern directions denoted `E`, `W`, `N` and `S` respectively. Additionally, the outer nodes of the mesh have wraparound links completing the torus, denoted `AE`, `AW`, etc. All channels are divided into two virtual channels, yielding two subnetworks. The virtual channel that is part of channel c belonging to subnetwork s is denoted c_s.

The clue routing algorithm is a highly adaptive minimal routing scheme. Algorithm 1 presents the routing logic. It takes as input the destination and the size of the torus. It bases its routing decisions on the following rules:

1. Route to channels in Subnetwork 1 whenever it provides a minimal route towards the destination (Line 3).
2. Route to channels $\mathtt{E}_2$, $\mathtt{W}_2$, $\mathtt{N}_2$, and $\mathtt{S}_2$ according to XY routing, only if the packet can reach its destination without using a wraparound link (Line 17).
3. Route to channels $\mathtt{AN}_2$ and $\mathtt{AS}_2$ only if the packet can reach its destination without using a horizontal wraparound link (Lines 10-13).
4. Route to channels $\mathtt{AE}_2$ and $\mathtt{AW}_2$ whenever if it provides a minimal route towards the destination (Lines 6-9).

Algorithm 1 The clue routing function.

Require: Coordinates (n_x, n_y) of end of current channel
Require: Coordinates (d_x, d_y) of destination
Require: Dimension (s_x, s_y) of torus
1: $\delta_x = d_x - n_x$;
2: $\delta_y = d_y - n_y$;
3: ret $+=\{c_1 \mid c_1$ provides a minimal route$\}$
4: **if** $|\delta_x| > \frac{1}{2}s_x \vee |\delta_y| > \frac{1}{2}s_y$ **then**
5: **if** at border **then**
6: **if** $\delta_x > \frac{1}{2}s_x$ **then**
7: ret $+= \mathtt{AW}_2$
8: **else if** $\delta_x < -\frac{1}{2}s_x$ **then**
9: ret $+= \mathtt{AE}_2$
10: **else if** $|\delta_x| \leq \frac{1}{2}s_x \wedge \delta_y > \frac{1}{2}s_y$ **then**
11: ret $+= \mathtt{AS}_2$
12: **else if** $|\delta_x| \leq \frac{1}{2}s_x \wedge \delta_y < -\frac{1}{2}s_y$ **then**
13: ret $+= \mathtt{AN}_2$
14: **end if**
15: **end if**
16: **else**
17: ret $+= c_2$ if c_2 provides an XY route
18: **end if**

Example 1. Consider a 7×7 torus (see Fig. 1). Let m be a message injected at node (1, 5) destined for (5, 1). The message has many different routes. It can first go north, take a vertical wraparound link, subsequently head west at node (1, 1), take the horizontal wraparound link and arrive at its destination. It can also take route r, drawn in bold in Fig. 1, by first going west, taking the horizontal wraparound link, subsequently heading north at node (5, 5), taking the vertical wraparound to arrive at its destination. Additionally, all routes ''in between'' these two routes are supplied, e.g., go west at (1, 5) and then go north.

We consider route r. Message m starts in Subnetwork 1, since by Rule 2 Subnetwork 2 cannot be used (the message still has to take a wraparound link). When m arrives at (0, 5), it can use both horizontal wraparound links (Rule 4). After using the wraparound link the message arrives at node (6, 5). Here it can go west or north, where Rule 2 ensures that both steps occur in Subnetwork 1. At node (5, 6) Rule 3 applies, ensuring that both vertical wraparound links to (5, 0) can be used. From (5, 0), no wraparound links are needed to arrive at the destination. From this node, Subnetwork 1 supplies all minimal routes, and Subnetwork 2 supplies XY routing.

For wormhole networks, clue causes deadlocks. Fig. 2 describes such a deadlock in a 7×7 torus. The deadlock consists of four worms using channels in Subnetwork 1. One worm occupying channels $(2, 6, \mathtt{W}_1), (1, 6, \mathtt{S}_1), (1, 5, \mathtt{S}_1)$ has destination (6, 4). As the shortest route requires the horizontal wraparound link, it is routed west towards $(1, 4, \mathtt{W}_1)$. Note that subnetwork 2 is not supplied, due to Rule 2. Channel $(1, 4, \mathtt{W}_1)$ is occupied by a worm routed north towards $(0, 4, \mathtt{N}_1)$. Again, because this worm still needs to traverse a wraparound link, Rule 2 ensures that this worm is not routed towards Subnetwork 2. A worm occupying channels $(0, 4, \mathtt{N}_1), (0, 5, \mathtt{E}_1), (1, 5, \mathtt{E}_1)$ is routed north as well. Its next hop is occupied by a fourth worm,

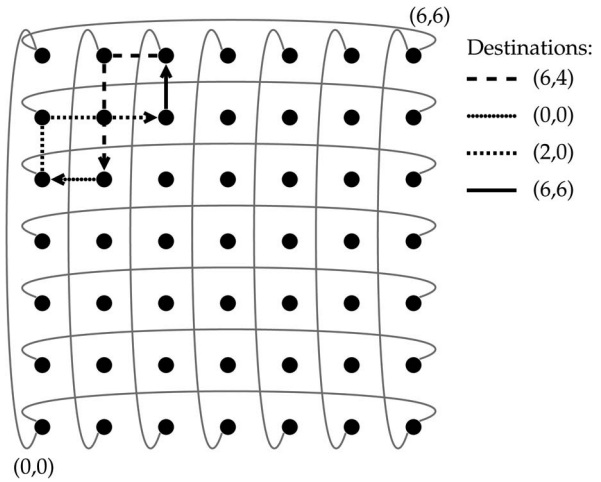


Fig. 2. Deadlock in a 2D Torus with clue routing.

which is routed towards channel $(2, 6, \mathtt{W}_1)$, completing the deadlock.

Once a worm reaches Subnetwork 2, clue provides deadlock-free XY routing. Any deadlock therefore consists of worms in Subnetwork 1 with destinations which route exclusively to Subnetwork 1. In the depicted deadlock, each worm still needs to traverse at least one wraparound link to reach its destination. Therefore, the next hops of all the worms are in Subnetwork 1 only.

This deadlock is actually the smallest deadlock possible. In Section 6, we prove that tori of sizes less than 7×7 are deadlock-free. For larger topologies, we prove that a deadlock involves at least four worms.

Luo and Xiang provide a deadlock-free solution called *wormhole clue*. Wormhole clue is similar to clue, with an added restriction on when channels in Subnetwork 1 are supplied. When a packet still needs to traverse one or more wraparound links, channels in Subnetwork 1 are supplied only in the dimensions in which the packet still needs to use these wraparound links.

4 Algorithm

This section introduces our approach for deciding deadlock freedom of wormhole networks. The first step decides a condition sufficient for the absence of deadlocks. In case Step 1 cannot prove deadlock freedom, the second step uses the output of the first step to decide a necessary and sufficient condition. We start by presenting a formal definition of deadlock and our necessary and sufficient condition.

4.1 Definitions

The network model presented here is conceptually equal to the models used in most literature [5], [2], [13] and is a straightforward formalization of a wormhole network without message dependencies. Note that our use of channels refers to either physical or virtual channels. For our theory, it is irrelevant whether a channel is physical or virtual.

Definition 1. *A network N is a tuple with a finite set of processing nodes P, a finite set of arcs C, a domain of flits F, a domain of message identifiers M, two topology functions* $\mathrm{src} : C \mapsto P$ *and* $\mathrm{end} : C \mapsto P$, *a capacity function* $\mathrm{cap} : C \mapsto \mathbb{N}$, *a function* $\mathrm{msg} : F \mapsto M$, *a function* $\mathrm{dest} : M \mapsto P$, *and a routing function* $\mathbf{R} : P \times P \mapsto \mathcal{P}(C)$

$$N \stackrel{\text{def}}{=} \langle P, C, F, M, \mathrm{src}, \mathrm{end}, \mathrm{cap}, \mathrm{msg}, \mathrm{dest}, \mathbf{R} \rangle.$$

The topology of network N is defined by functions $\mathrm{src} : C \mapsto P$ and $\mathrm{end} : C \mapsto P$. These functions return the processing nodes at the source and at the end of a channel. The routing function computes from each processing node s and each destination node d a set of next hops $\mathbf{R}(s, d)$. This set is nonempty if $s \neq d$. Each channel c has a certain number of places, where each place can either be empty or store exactly one flit. An empty place is denoted with ϵ. We assume $\epsilon \notin F$ and let $F_\epsilon = F \cup \{\epsilon\}$ denote the domain of flits plus the empty place. The capacity of channel c, denoted $\mathrm{cap}(c)$, is defined as the number of places of channel c.

Flits are the atomic units of transfer that are transmitted through channels, i.e., they are concrete pieces of data that are to be communicated. At any time, channels may contain many nonunique flits. Each flit belongs to at most one message, but each message may consist of multiple flits. In our definitions and proofs, we need to be able to distinguish equal flits belonging to different messages. To this end, the set of unique message identifiers M is used. Function msg is able to retrieve, given a flit f, the unique message id of the message to which flit f belongs. Since flits are not unique, some additional bookkeeping is required for the actual implementation of this function. Finally, function dest returns the destination of the message corresponding to the given message identifier.

For sake of presentation, we introduce some shorthand notations. Let f be a flit, let list L be a list with elements in F_ϵ, and let c_i denote a channel.

- The destination of a flit f:
$$\mathrm{dest}(f) = \mathrm{dest}(\mathrm{msg}(f)).$$
- The set of destinations that route from c_0 to c_1:
$$\Delta(c_0, c_1) = \{d \in P \mid c_1 \in \mathbf{R}(c_0, d)\}.$$
- The set of messages of a list of flits L:
$$\mathrm{msgs}(L) = \{m \in M \mid \exists f \in L - \epsilon \cdot \mathrm{msg}(f) = m\}.$$
- All the next hops of c_0:
$$\mathbf{R}(c_0) = \{c_1 \in C \mid \exists d \in P \cdot c_1 \in \mathbf{R}(c_0, d)\}.$$

Since channels store messages and there are no further state holding components in the network, the state of the network is determined completely by the state of the channels. A state or configuration contains information on which flits are in which places.

Definition 2. *A configuration σ is an assignment of lists of flits or empty places to channels*

$$\sigma : C \mapsto \mathcal{L}(F_\epsilon).$$

Function σ returns a *list* of flits. The list of flits in channel c can be accessed through $\sigma(c)$. This list contains the flit stored in the head of the channel followed by flits stored in the tail of the channel. Given a channel c, the flit which

occupies the head of the channel is returned by $\sigma(c)[0]$. The number of flits in c is denoted $|\sigma(c)|$. The empty configuration is denoted by σ_ϵ. Given a configuration σ and a message id m, it is possible to compute the set of channels that is occupied by the message corresponding to message id m. This set of channels is denoted by $\mathrm{occ}(m, \sigma)$.

Not any assignment of flits to channels constitutes a legal configuration. First, no channel capacities may be exceeded. Secondly, for sake of presentation we assume the traditional variant of wormhole switching allows messages to acquire a resource only when it is completely empty. Thus, each channel can contain flits belonging to one message only. In Appendix D, which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.121, we prove that deadlock freedom of wormhole switching with nonatomic channel allocation (such as in [22]) is logically equivalent to deadlock freedom of traditional wormhole switching. Therefore, our tool applies equally to the non atomic case. Thirdly, worms occupy paths of channels in the network. To this end we introduce predicate $\mathbf{R}\mathrm{path}$ which given a set of channels π and a destination d returns true if and only if π is a path of channels that can be established by the routing function for destination d. The channels occupied by a worm always constitute such a path. We denote a set of routing paths $\{\pi^{d_0}, \pi^{d_1} \ldots, \pi^{d_k}\}$ with $\Pi^\star$.

Definition 3. *A configuration σ is legal, notation $\mathrm{legal}(\sigma)$, if and only if for any channel the capacity is not exceeded, all flits belong to the same message, and all flits can be routed towards their current channel*

$$\mathrm{legal}(\sigma) \stackrel{\mathrm{def}}{=} \forall c \in C \begin{cases} |\sigma(c)| = \mathrm{cap}(c) \\ |\mathrm{msgs}(\sigma(c))| \leq 1 \\ \forall m \in \mathrm{msgs}(\sigma(c)) \cdot \mathbf{R}\mathrm{path}(\mathrm{occ}(m, \sigma), \mathrm{dest}(m)). \end{cases}$$

Note that the first conjunct states that a configuration always assigns exactly $\mathrm{cap}(c)$ elements in F_ϵ to channel c. In case channel c is not full, the configuration assigns empty places to the channel.

We informally define *reachability*. A formal definition requires the formalization of the behavior of wormhole networks, e.g., when messages are injected and how they move around. For sake of readability and space limitation, we omit all the corresponding formal definitions.

Definition 4. *A configuration σ is reachable, notation $\mathrm{reachable}(\sigma)$, if and only if there exists an execution of the network starting in the empty configuration and ending in σ.*

A deadlock configuration is required to be legal and reachable. It must be nonempty. Additionally, all flits must be blocked. A header flit is blocked if all its next hops contain at least one flit. A tail flit is blocked if the next channel of the worm is full. We assume predicates hd and tl are available which return true if and only if the given flit is a header (tail) flit. Function $\mathrm{next} : F \times \boldsymbol{\Sigma} \mapsto C$ returns the next channel of a tail flit given the current configuration.

Definition 5. *For wormhole network N, a configuration σ is a deadlock configuration, notation $\boldsymbol{\Omega}(\sigma)$, if and only if it is legal, reachable, nonempty, no header flit has arrived at its destination and all flits are blocked*

$$\begin{aligned} \boldsymbol{\Omega}(\sigma) & \\ & \stackrel{\mathrm{def}}{=} \mathrm{legal}(\sigma) \\ & \wedge \mathrm{reachable}(\sigma) \\ & \wedge \sigma \neq \sigma_\epsilon \\ & \wedge \forall c \in C \cdot \forall f \in \sigma(c) \cdot \mathrm{hd}(f) \Longrightarrow \\ & \qquad \begin{cases} \mathrm{dest}(f) \neq \mathrm{end}(c) \\ \forall n \in \mathbf{R}(\mathrm{end}(c), \mathrm{dest}(f)) \cdot |\sigma(n) - \epsilon| > 0 \end{cases} \\ & \wedge \forall c \in C \cdot \forall f \in \sigma(c) \cdot \mathrm{tl}(f) \Longrightarrow \epsilon \notin \sigma(\mathrm{next}(f)). \end{aligned}$$

4.2 Complete Condition

We have proven a necessary and sufficient condition for deadlock-free routing in wormhole networks [5]. In a deadlock, all worms are blocked. As flits in the tail always follow the header flit, blockage of a message depends solely on the header flit. The central idea of the condition is that a header flit must always be at an *escape*, i.e., a channel from which there will always be an available next hop. The set of worms in a wormhole configuration is represented by a set of routing paths. As worms cannot intersect, the set of worms of a legal configuration always corresponds to a pairwise disjoint set of routing paths.

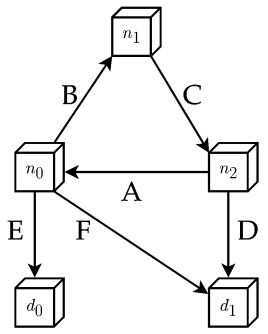
Consider the configuration in Fig. 2. Four messages occupy four paths of channels. For each routing path, the head of the path cannot escape as the routing function does not supply next hops that are not included in the set of paths. There exists a set of paths without an escape, which corresponds to the existence of a deadlock.

Now consider the following change to the routing function. Instead of using Subnetwork 2 only when no wraparound links are needed to get to the destination, the routing logic always supplies a next hop of Subnetwork 2 (Line 17 in Algorithm 1 is moved after Line 3). With this routing logic, the configuration depicted in Fig. 2 is not a deadlock any more. As there is at least one worm that has a next hop not included in the set of paths, the header flit of that worm can move towards this escape. This breaks the circular wait. The set of next hops has an escape and is therefore not a deadlock.

Definition 6. *Given a set of routing paths $\Pi^\star$, a channel e is an escape for $\Pi^\star$, notation $\mathrm{esc}(e, \Pi^\star)$, if and only if channel e is the head of some path and either e is a sink or e has a next hop that is not contained in $\Pi^\star$*

$$\begin{aligned} \mathrm{esc}(e, \Pi^\star) & \stackrel{\mathrm{def}}{=} \exists \pi^d \in \Pi^\star \cdot e \\ & = \pi^d[0] \wedge \left(\mathbf{R}(e, d) = \emptyset \vee \mathbf{R}(e, d) \not\subseteq \bigcup \Pi^\star \right). \end{aligned}$$

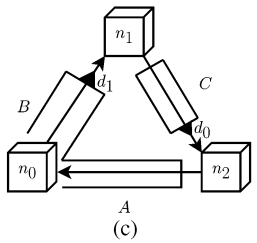
Our condition states that a network is deadlock-free if and only if any pairwise disjoint set of routing paths has an escape, or contrapositively:



(a)

n	$\mathbf{R}(n, d_0)$	$\mathbf{R}(n, d_1)$
n_0	E	B, F
n_1	C	C
n_2	A	A, D

(b)



(c)

c	mark(c)	class(c)
A	attain	$d_1 \mid d_0 d_1$
B	sens	$d_1 \mid \_$
C	sens	$d_0 d_1 \mid d_1$
D	immune	$\_ \mid d_1$
E	immune	$\_ \mid d_0$
F	immune	$\_ \mid d_1$

(d)

Fig. 3. Example of deadlock. (a) Interconnection network. (b) Routing function. (c) Deadlock. (d) Result of Step I.

Theorem 1. *A wormhole network has a deadlock if and only if there exists a pairwise disjoint set of routing paths without an escape*

$$\exists \sigma \in \boldsymbol{\Sigma} \cdot \boldsymbol{\Omega}(\sigma) \Longleftrightarrow \exists \Pi^{\star} \cdot \begin{cases} \Pi^{\star} \neq \emptyset & (A) \\ \bigcap \Pi^{\star} = \emptyset & (B) \\ \forall c \in C \cdot \neg \mathrm{esc}(c, \Pi^{\star}) \,. & (C) \end{cases}$$

Property (A) states that the set of paths is nonempty. The paths must be pairwise disjoint, which is enforced by Property (B). Property (C) states that the set of paths has no escape.

We will overload symbol $\boldsymbol{\Omega}$. When applied to a configuration σ, it returns true if and only if the configuration is a deadlock (see Definition 5). If applied to a set of routing paths $\Pi^{\star}$, it will return true if and only if the three properties of Theorem 1 hold for the given set of routing paths.

4.3 Step I: Checking a Sufficient Condition

The first step of the algorithm decides a condition similar to Theorem 1 but without Property (B). This condition is sufficient for deadlock freedom. If Step I concludes that there is no deadlock, this result is sound. Otherwise, a possible deadlock is returned, which is passed to Step II for further analysis.

4.3.1 Overview

The basic objective of Step I is to mark each channel as *deadlock-immune*, *deadlock-sensitive* or *deadlock-attainable*. The intuition behind the first marking is that in a deadlock-immune channel no packet can be permanently blocked. In a deadlock-free network, all channels are deadlock-immune. In a deadlock-sensitive channel, it is possible for header- or tail flits to be permanently blocked. Finally, a channel is deadlock-attainable if tail flits can be permanently blocked in that channel, but header flits cannot. After termination of the algorithm, the markings are used to either serve as input for Step II, or to state that the network is deadlock-free.

To determine the marking of a channel, the algorithm classifies all destinations for which a message can reach that channel. A classification $x \mid y$ of channel c stands respectively for destinations that may lead packets from channel c into a circular wait and for destinations for which channel c is an escape. As routing is adaptive, a destination may be classified as both.

Let c be a channel with classification $x \mid y$. First, consider the case where x is not a subset of y. Apparently, there exists some destination d which leads into a circular wait, and for which there is no escape out of these cycles. Consequently, a flit can be permanently blocked in channel c. The channel is marked deadlock-sensitive. Now consider the case where x is a subset of y. This means that all destinations that lead into a circular wait, also provide escapes out of these cycles. Consequently, no header flit can be permanently blocked in channel c. If there exists a routing path towards some deadlock-sensitive channel h, it is still possible for tail flits to be permanently blocked. This can happen if the routing path is filled with a worm with its header flit in h. In this case, the channel is marked deadlock-attainable. If no such path exists, it is not possible for any flit to be permanently blocked and the channel is marked deadlock-immune.

Example 2. Consider the network in Fig. 3a. The network consists of five processing nodes. For sake of clarity, only the processing nodes d_0 and d_1 are destination nodes. The routing function is depicted in Fig. 3b. There is exactly one deadlock possible. This deadlock has two worms (see Fig. 3c). One worm occupies channels A and B. Its tail is in A and its header flit is in B. One worm occupies channel C only and is destined for d_0. The worms wait for each other and have no alternative routes towards their destinations. The set of routing paths $\{[B, A], [C]\}$ has no escape. There is a deadlock.

Fig. 3d shows the result of Step I. Channels B and C—the heads of the worms—have been marked deadlock-sensitive. Channel A—the tail of a worm—is marked deadlock-attainable. The other channels are deadlock-immune.

4.3.2 Pseudocode

Algorithm 2 (see Appendix A) shows pseudo code for checking a condition sufficient for deadlock freedom. This part is basically a depth-first search, keeping track of visited channels to deal with cycles. Effectively, this yields a spanning tree. When a bottom leaf has been reached, there are two possibilities: either a cycle or a sink has been reached. In the first case, information is propagated upwards to indicate that this part of the tree leads to a circular wait. In the second case, information is propagated upwards to indicate that this part of the tree leads to an escape.

Algorithm 3 executes CREATETREE for all unmarked channels c, with $C_I = \mathbf{R}(c)$ and $p = c$. It returns the—possibly empty—set of deadlock-sensitive and deadlock-attainable channels.

Algorithm 3 returns any channel that can possibly participate in a deadlock. Filling all the channels with tail- and header flits yields a configuration which satisfies Properties (A) and (C) of Theorem 1. It remains to be answered whether it is possible to fill the channels in such a way that a legal deadlock is obtained, where worms do not intersect.

Also, the current intermediate result contains *any* channel that can possibly participate in a deadlock. Such a maximal deadlock is often not very readable, and does often not provide clear insight into the actual cause of deadlocks. Step II of our algorithm uses the result obtained by STEP I to find a minimal and legal set of routing paths that satisfies all properties of Theorem 1.

Example 3. When applying STEP I to the torus with clue routing as presented in Section 3, it marks all the channels of Subnetwork 2 deadlock-immune. These channels cannot participate in any deadlock. In Subnetwork 1, most channels are marked as either deadlock-sensitive or deadlock-attainable. The destinations that prevent the channels from being deadlock-immune are those destinations that route towards Subnetwork 1. Further analysis is required to determine whether the channels in Subnetwork 1 can be filled with a legal and reachable deadlock.

4.4 Step II: Checking a Complete Condition

We provide a reduction from the deadlock decision problem to an integer linear programming instance (ILP). Task of this reduction is to find a pairwise disjoint set of routing paths that has no escape. As a result, a solution of the ILP corresponds to a deadlock. If no solution exists, the network is deadlock-free.

First, we use the set of sens- and attain-marked channels to enumerate all paths that can possibly be in such a set. This creates an enumeration $\Pi^\star_{\mathtt{E}} = \pi_0^{d_0}, \pi_1^{d_1}, \ldots, \pi_I^{d_I}$ of I routing paths. Formally, this enumeration is defined as the smallest set that satisfies the following constraint:

$$\forall \pi^d \cdot \begin{cases} & \mathbf{R}\mathrm{path}(\pi^d, d) \\ \wedge & \mathrm{mark}\big(\pi^d[0]\big) = \mathrm{sens} \\ \wedge & \forall c \in \pi^d \cdot \mathrm{mark}(c) \in \{\mathrm{sens}, \mathrm{attain}\} \end{cases} \Longrightarrow \pi^d \in \Pi^\star_{\mathtt{E}}.$$

Any routing path that consists of sens- and attain-marked channels and whose head is marked sens is a member of the enumeration. We assume that the set of channels can be enumerated as well, yielding an enumeration $C_{\mathtt{E}} = C_0, C_1, \ldots, C_J$ where $J = |C| - 1$.

In the ILP, we introduce one Boolean variable p_i for each path and one Boolean variable c_j for each channel. Intuitively, a variable is true if and only if it is part of the deadlock. A variable p_i is true if and only if a worm participates in the deadlock occupying exactly the channels of path π_i, destined for d_i. A variable c_j is true if and only if channel C_j is nonempty in the deadlock. Given the enumerations $\Pi^\star_{\mathtt{E}}$, we generate the following ILP:

$$\text{minimize}$$

$$\sum_{i=0}^{I} p_i \tag{O}$$

$$\text{subject to}$$

$$\sum \big\{ c_j \mid \exists i \cdot C_j \in \pi_i^{d_i} \big\} \geq 1 \tag{A}$$

$$-c_j + \sum \big\{ p_i \mid C_j \in \pi_i^{d_i} \big\} = 0 \quad : 0 \leq j \leq J \tag{B}$$

$$-p_i + c_j \geq 0 \quad : \begin{matrix} 0 \leq i \leq I \\ C_j \in \mathbf{R}\big(\pi_i^{d_i}[0], d_i\big). \end{matrix} \tag{C}$$

We denote the ILP derived from the enumeration of routing paths $\Pi^\star_{\mathtt{E}}$ with $\mathrm{ILP}(\Pi^\star_{\mathtt{E}})$.

The objective of the integer linear program is the minimization of the number of paths, and as a result the number of worms. The constraints of the ILP directly correspond to the properties stated in the necessary and sufficient condition of Theorem 1. Constraint (A) formalizes nonemptiness. All variables c_j are summed, for which the corresponding channel C_j is member of one of the paths in the enumeration. This sum must at least be one, as at least one channel must be filled for a deadlock to occur. Constraints (B) enforce that the solution of the ILP consists of pairwise disjoint paths. For any channel C_j, we sum the variables p_i for which the corresponding path $\pi_i^{d_i}$ contains channel C_j. We subtract the variable c_j. This enforces that *if* channel C_j is chosen, then at most one path that contains channel C_j is chosen. By requiring the constraint to be equal to zero, we additionally enforce that if channel C_j is not chosen, no path that contains channel C_j is chosen. Constraints (C) enforce that a solution of the ILP has no escape. For any path $\pi_i^{d_i}$ and for any next hop C_j of that path, we enforce that *if* the path is chosen, the next hop is chosen as well.

Example 4. Consider again the deadlock described in Fig. 2. The deadlock consists of four worms. Constraint (A) sums the Boolean variable c_j corresponding to each channel that is member of one of these paths. As this value is at least one, there is at least one channel participating in the deadlock.

For each channel there is one constraint of type (B). Let path $\pi_0^{d_0} = [(1,5,\mathtt{S}_1),(1,6,\mathtt{S}_1),(2,6,\mathtt{W}_1)]^{(6,4)}$ be the routing path corresponding to Worm 1. We consider the constraint for channel $(2,6,\mathtt{W}_1)$, which is the nth channel in enumeration $C_{\mathtt{E}}$:

$$-c_n + p_0 + \cdots = 0.$$

If variable c_n is true, channel $(2,6,\mathtt{W}_1)$ contains flits. Exactly one of the paths containing this channel has to be chosen as well. Path $\pi_0^{d_0}$ is one of them. If c_n is false, variable p_0 is false as well, since channel $(2,6,\mathtt{W}_1)$ does not contain flits and therefore path $\pi_0^{d_0}$ cannot participate in the deadlock.

For each next hop of the head of each path, there is a constraint of type (C). Worm 1 in Fig. 2 has one next hop, namely channel $(1,4,\mathtt{W}_1)$. Let this channel be the mth channel in enumeration $C_{\mathtt{E}}$. The constraint that has to be met is:

$$-p_0 + c_m \geq 0.$$

If variable p_0 is true, path $\pi_0^{d_0}$ participates in the deadlock, representing that Worm 1 participates in the deadlock. The constraint enforces that p_0 implies that the next hop of this worm, channel $(1,4,\mathtt{W}_1)$ is nonempty.

We supply the ILP to an off-the-shelf linear programming solver. If a solution is found, a deadlock configuration is returned by filling—for all i such that p_i is true in the solution—the routing path $\pi_i^{d_i}$ with a worm destined for d_i. If no solution is found, the network is deadlock-free.

The ILP generated from the set of routing paths considers two facets of each routing path: which channel it occupies and to which next hops the worm corresponding to the path is routed. It is possible that two paths consist of the same channels, and have the same next hops, but with a different destination. If this is the case, it is not necessary for both paths to be in the enumeration. We define an equivalence

relation over routing paths. The purpose of this relation is that if two paths are in the same equivalence class, one of these paths can be removed from the enumeration since they will both yield equal results.

Definition 7. *Two routing paths π^{d_0} and π^{d_1} are equivalent, notation $\pi^{d_0} \equiv \pi^{d_1}$, if and only if they consist of the same channels and have the same next hops*

$$\pi^{d_0} \equiv \pi^{d_1} \stackrel{\text{def}}{=} \begin{cases} & |\pi^{d_0}| = |\pi^{d_1}| \\ \wedge & \forall i < |\pi^{d_0}| \cdot \pi^{d_0}[i] = \pi^{d_1}[i] \\ \wedge & \mathbf{R}\big(\pi^{d_0}[0], d_0\big) = \mathbf{R}\big(\pi^{d_1}[0], d_1\big). \end{cases}$$

Instead of using the enumeration $\Pi^{\star}_{\mathrm{E}}$ to generate the ILP, we use its quotient set $\Pi^{\star}_{\mathrm{E}}/\equiv$, where each equivalence class is represented by one canonical element.

Algorithm 4 shows the pseudo code of Step II of our approach. Combined, Steps I and II decide deadlock freedom of wormhole networks, i.e., they check whether the necessary and sufficient condition of Theorem 1 holds.

5 Theorems

We prove correctness of our algorithm. The lemma's and theorems are stated here, their proofs can be found in Appendix B. First, we prove that the deadlock-immune mark given by Step I to channels is always sound. This means that once a channel is marked deadlock-immune, it is not possible for the channel to participate in any deadlock, i.e., no flit can be permanently blocked in the channel. Effectively, this proves sufficiency of Step I, i.e., if Step I returns the empty set then the network is deadlock-free. Lemma 1 has been mechanically proven correct, using the ACL2 theorem prover.

Lemma 1. *If a channel c is marked immune, the channel is not a member of a set of routing paths $\Pi^{\star}$ without an escape*

$$\mathrm{mark}(c) = \mathrm{immune} \wedge \boldsymbol{\Omega}(\Pi^{\star}) \Longrightarrow c \notin \bigcup \Pi^{\star}.$$

Lemma 1 concerns deadlock-immune channels. Ideally, we want to prove a similar lemma for deadlock-sensitive channels as well. This is not possible, as this marking is not stable, i.e., Step II can show that a channel has been incorrectly marked deadlock-sensitive by Step I. In other words, a deadlock-sensitive mark after Step I does not give us any certain information.

If a channel c is marked attain, this *does* give us some certain information. We do not know, with certainty, that tail flits can be permanently blocked in channel c. But we do know that in channel c a header flit can never be permanently blocked. Before proving correctness of our algorithm, we first prove a lemma concerning deadlock-attainable marks. This lemma has also been proven mechanically using the ACL2 theorem prover.

Lemma 2. *If a channel c is marked attain, the channel is not at the head of a set of routing paths $\Pi^{\star}$ without an escape*

$$\mathrm{mark}(c) = \mathrm{attain} \wedge \boldsymbol{\Omega}(\Pi^{\star}) \Longrightarrow \quad \forall \pi^{d} \in \Pi^{\star} \cdot \pi^{d}[0] \neq c.$$

Using these lemma's, we can prove the final theorem. We show that if a solution is found for the ILP derived from the markings of Step I, this solution corresponds to a deadlock. Also, we show that any deadlock necessarily consists of paths from $\Pi^{\star}_{\mathrm{E}}$, which implies that if the ILP is infeasible, the network is deadlock-free.

A solution to an ILP is an assignment of values to the variables occurring in the ILP. Let ϕ be such a function. We let $\mathrm{sol}(\phi, I)$ return true if and only if ϕ is a solution for ILP I.

Theorem 2. *The ILP derived from the enumeration of routing paths $\Pi^{\star}_{\mathrm{E}}$ based on the markings of Step I is feasible, if and only if there exists a set of routing paths without an escape*

$$\exists \phi \cdot \mathrm{sol}\big(\phi, \mathrm{ILP}\big(\Pi^{\star}_{\mathrm{E}}\big)\big) \Longleftrightarrow \exists \Pi^{\star} \cdot \boldsymbol{\Omega}(\Pi^{\star}).$$

6 Applications

We have incorporated the algorithm in the tool DCI2. DCI2 adds various features to the algorithm that are necessary to make them valuable in practice. DCI2 takes as input a description of the topology and a C++ implementation of the routing function. Our tool has the following features:

- The tool checks whether the model of the routing function is correct with respect to the topology. In the hardware it is impossible for a packet to be routed through channels that do not exist. The model written in C++ code however can route, e.g., a packet west at the left-most column, or route diagonally even when a diagonal channel does not exist. The tool detects such errors and outputs them.
- The tool checks whether the routing function is *connected*. When routing logic is erroneously implemented, it is possible that a packet arriving at a processing node has no path to its destination. If this is the case, the tool outputs exactly which packet in which channel with which destination has no next hops.
- The tool detects *livelocks*. Livelocks can be hard to find manually. The tool checks whether scenarios exist in which one packet gets into a cycle. If a livelock has been found, we report the exact trace leading to this cycle and the cycle itself.

We have applied DCI2 to the clue routing logic presented in Section 3. These results have been obtained on a Sun Fire X4440 machine, with four 2.3 GHz Quad-Core AMD Opteron 8356 processors (16 cores in total) and 128 GB of memory. We have used CPLEX to solve integer linear programming problems.

The minimal deadlock in Fig. 2 has been found by our tool within 3 seconds. Finding minimal deadlocks is a computationally very hard problem. For the clue routing logic, our method scales up to a 9×9 torus. In practice, a minimal deadlock is found relatively quickly. Proving that this deadlock actually is minimal is the most expensive computation task. For larger meshes, we are still able to find deadlocks but proving their minimality becomes intractable. In summary, our method scales up in finding small and readable deadlocks, but not the theoretical minimal ones.

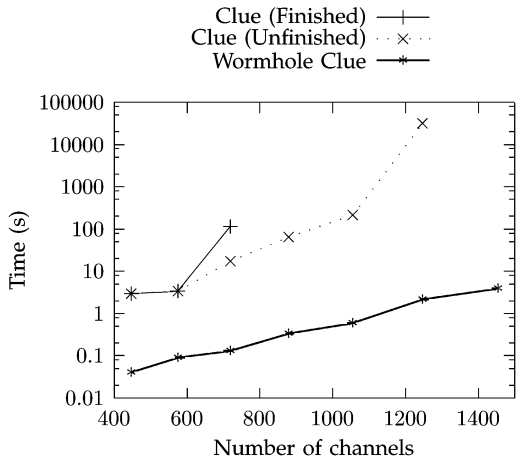


Fig. 4. Experimental results on clue routing logic.

Fig. 4 shows experimental results. Curve ''Finished'' depicts the running times of our algorithm without any interference. Curve ''Unfinished'' has been obtained by stopping the algorithm as soon as a deadlock of four worms has been found. In an 11×11 torus with 1056 channels, we can find a deadlock of four worms within 4 minutes. In a 12×12 torus this already takes about 9 hours. A deadlock with only 5 worms is found, however, within 4 hours.

We have also run our tool on wormhole clue, which is deadlock-free. This yields significantly better running times. Our algorithm proves absence of deadlocks for a 13×13 2D torus with 1456 channels within 4 seconds.

For the clue routing logic, Step 1 of our algorithm returns a possible deadlock, which is then confirmed to be legal and reachable by Step 2. We could only find very academic and contrived examples of routing functions where Step 1 suspects a deadlock, which is then ruled out by Step 2. We presented such an example previously [3]. The example network is deadlock-free, but could have deadlocks if worms could intersect. The corresponding ILP is proven infeasible within a second.

For more experimental results, we refer to Appendix C.

7 Conclusion

Wormhole switching is a complex technique with many subtleties. Designing adaptive routing functions for wormhole networks is therefore a difficult task. It often involves a proof that the routing logic satisfies a complex condition for deadlock-free routing. We provided the first algorithm which automatically decides absence of deadlocks in wormhole networks. It provides readable feedback in case the network is not deadlock-free. Our algorithm is sound and complete.

The correctness of our condition and algorithm has been formally checked using the ACL2 interactive theorem proving system. The proof effort involves 88 definitions and 632 theorems for a total of 7263 lines of code. We implemented our algorithm in C++ and experimental results confirm that our algorithm scales sufficiently even for hard cases such as the clue routing logic.

Our deadlock detection algorithm is incorporated in the tool DCI2. Besides deadlocks, DCI2 checks for livelocks, and various minor errors in the routing logic. As DCI2 is able to provide readable feedback quickly, it is applicable to any combination of topology and routing logic, and requires little input, it can be used as a formal debugging tool during the design of interconnects and routing functions. This opens the way to new design methodologies. One can design initial routing logic based on, e.g., performance considerations, analyze the routing logic using DCI2, and modify the routing logic based on this analysis. This way of designing routing logic has been explored in [21].

The most significant and actual application of our fast algorithm is probably the verification of fault-tolerant routing functions. Such a function should be deadlock-free for all possible faults that can occur. An interesting future work would study how—given a routing function—our algorithm performs in the deadlock analysis for different positions of faulty routers.

Acknowledgment

The authors would like to thank Z. Yu from the Tsinghua University for suggesting the clue routing logic as an interesting application of our tool and his cooperation with obtaining these results. This research is supported by NWO/EW project Formal Validation of Deadlock Avoidance Mechanisms (FVDAM) under Grant 612.064.811. This research is supported by a grant from Intel Corporation.

References

[1] W.J. Dally and C. Seitz, ''Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,'' *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547-553, May 1987.

[2] J. Duato, ''A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 10, pp. 1055-1067, Oct. 1995.

[3] F. Verbeek and J. Schmaltz, ''A Comment on 'A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks','' *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1775-1776, Oct. 2011.

[4] C.J. Glass and L.M. Ni, ''The Turn Model for Adaptive Routing,'' *J. ACM*, vol. 41, no. 5, pp. 874-902, Sept. 1994.

[5] F. Verbeek and J. Schmaltz, ''On Necessary and Sufficient Conditions for Deadlock-Free Routing in Wormhole Networks,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 12, pp. 2022-2032, Dec. 2011.

[6] W. Luo and D. Xiang, ''An Efficient Adaptive Deadlock-Free Routing Algorithm for Torus Networks,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 800-808, May 2012.

[7] F. Verbeek and J. Schmaltz, ''Automatic Verification for Deadlock in Networks-on-Chips with Adaptive Routing and Wormhole Switching,'' in *Proc. IEEE NOCS*, May 2011, pp. 25-32.

[8] M. Kaufmann, P. Manolios, and J.S. Moore, *ACL2 Computer-Aided Reasoning: An Approach*. Dordrecht, The Netherlands: Kluwer, 2000.

[9] F. Verbeek and J. Schmaltz, ''Formal Verification of a Deadlock Detection Algorithm,'' in *Proc. 10th Int. Workshop ACL2 Theorem Prover Appl.*, 2011, pp. 103-112.

[10] J. Duato, ''A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 12, pp. 1320-1331, Dec. 1993.

[11] L. Schwiebert and D. Jayasimha, ''A Necessary and Sufficient Condition for Deadock-Free Wormhole Routing,'' *J. Parallel Distrib. Comput.*, vol. 32, no. 1, pp. 103-117, Jan. 1996.

[12] E. Fleury and P. Fraigniaud, ''A General Theory for Deadlock Avoidance in Wormhole-Routed Networks,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 7, pp. 626-638, July 1998.

[13] S. Taktak, E. Encrenaz, and J.-L. Desbarbieux, ''A Polynomial Algorithm to Prove Deadlock-Freeness of Wormhole Networks,'' in *Proc. 18th Euromicro Int. Conf. PDP Netw.-Based Comput.*, Feb. 2010, pp. 121-128.
[14] J. Duato, ''A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Cut-Through and Store-and-Forward Networks,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 8, pp. 841-1067, Aug. 1996.
[15] F. Silla, M.P. Malumbres, A. Robles, P. López, and J. Duato, ''Efficient Adaptive Routing in Networks of Workstations with Irregular Topology,'' in *Proc. 1st Int. Workshop CANPC*, 1997, pp. 46-60.
[16] G.-M. Chiu, ''The Odd-Even Turn Model for Adaptive Routing,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 7, pp. 729-738, July 2000.
[17] D. Starobinski, M. Karpovsky, and L.A. Zakrevski, ''Application of Network Calculus to General Topologies using Turn-Prohibition,'' *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, pp. 411-421, June 2003.
[18] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, ''A Methodology for Design of Application Specific Deadlock-Free Routing Algorithms for NoC Systems,'' in *Proc. 4th Int. CODES+ISSS*, 2006, pp. 142-147.
[19] J. Duato, O. Lysne, R. Pang, and T. Pinkston, ''Part I: A Theory for Deadlock-Free Dynamic Network Reconfiguration,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 5, pp. 412-427, May 2005.
[20] O. Lysne, T.M. Pinkston, and J. Duato, ''Part II: A Methodology for Developing Deadlock-Free Dynamic Network Reconfiguration Processes,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 5, pp. 428-443, May 2005.
[21] A. Alhussien, F. Verbeek, N. Bagherzadeh, J. Schmaltz, and B. van Gastel, ''A Formally Verified Deadlock-Free Routing Function in a Fault-Tolerant NoC Architecture,'' in *Proc. 25th SBCCI*, 2012, pp. 1-6.
[22] S. Ma, N.E. Jerger, and Z. Wang, ''Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-Chip,'' in *Proc. 18th IEEE Int. Symp. HPCA*, 2012, pp. 467-478.
[23] L. Ni and P. McKinley, ''A Survey of Wormhole Routing Techniques in Direct Networks,'' *Computer*, vol. 26, no. 2, pp. 62-76, Feb. 1993.
[24] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, ''Hermes: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip,'' *Integration, VLSI J.*, vol. 38, no. 1, pp. 69-93, Oct. 2004.
[25] M. Coppola, S. Curaba, M. Grammatikakis, G. Maruccia, and F. Papariello, ''OCCN: A Network-on-Chip Modeling and Simulation Framework,'' in *Proc. DATE Conf.*, 2004, pp. 174-179.
[26] M. Coppola, M. Grammatikakis, R. Locatelli, G. Mariuccia, and L. Pieralisi, *Design of Interconnect Processing Units Spidergon STNoC*. Boca Raton, FL, USA: CRC Press, 2009.
[27] F. Karim, A. Nguyen, and S. Dey, ''An Interconnect Architecture for Networking Systems on Chips,'' *IEEE Micro*, vol. 22, no. 5, pp. 36-45, Sept./Oct. 2002.

Freek Verbeek received the MS and PhD degrees in computer science from the Radboud University Nijmegen, Nijmegen, Netherlands, in 2008 and 2013, respectively. Currently, he is an Assistant Professor at the Open University of the Netherlands. His research interests include interconnection networks and formal verification. His current work aims at using theorem proving techniques to prove security properties over separation kernels.

Julien Schmaltz received the PhD degree in electrical engineering from the University of Grenoble, Grenoble, France in 2006. Currently, he is an Assistant Professor at the Department of Mathematics and Computer Science of the Eindhoven University of Technology. His research interests include mechanized theorem proving and formal verification of multiprocessor systems-on-chip, with an emphasis on networks-on-chip.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.