Deadlock-Free Oblivious Routing for Arbitrary Topologies

Jens Domke
Center for Information Services and High Performance Computing
Technische Universität Dresden
Dresden, Germany
jens.domke@zih.tu-dresden.de

Torsten Hoefler
Blue Waters Directorate
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
htor@illinois.edu

Wolfgang E. Nagel
Center for Information Services and High Performance Computing
Technische Universität Dresden
Dresden, Germany
wolfgang.nagel@tu-dresden.de

***Abstract*—Efficient deadlock-free routing strategies are crucial to the performance of large-scale computing systems. There are many methods but it remains a challenge to achieve lowest latency and highest bandwidth for irregular or unstructured high-performance networks. We investigate a novel routing strategy based on the single-source-shortest-path routing algorithm and extend it to use virtual channels to guarantee deadlock-freedom. We show that this algorithm achieves minimal latency and high bandwidth with only a low number of virtual channels and can be implemented in practice. We demonstrate that the problem of finding the minimal number of virtual channels needed to route a general network deadlock-free is NP-complete and we propose different heuristics to solve the problem. We implement all proposed algorithms in the Open Subnet Manager of InfiniBand and compare the number of needed virtual channels and the bandwidths of multiple real and artificial network topologies which are established in practice. Our approach allows to use the existing virtual channels more effectively to guarantee deadlock-freedom and increase the effective bandwidth of up to a factor of two. Application benchmarks show an improvement of up to 95%. Our routing scheme is not limited to InfiniBand but can be deployed on existing InfiniBand installations to increase network performance transparently without modifications to the user applications.**

***Keywords*-acyclic path partitioning; deadlock-free; InfiniBand; NP-complete; routing; virtual channels**

I. Introduction

The number of network endpoints in supercomputer networks is growing steadily. It ranges from mid-range systems with 500–1000 endpoints over high-class systems such as Sandia's Thunderbird cluster with ≈ 4400 endpoints or the Ranger system at the Texas Advanced Computing Center with ≈ 4000 endpoints to top-class systems such as Oak Ridge's Jaguar (XT-5) system with ≈ 19000 endpoints or the soon-to-be installed Blue Waters system with ≈ 10000 endpoints. Hundreds of thousands of endpoints are considered in the design of Exascale systems [1].

It is important to exploit the performance of such large-scale networks efficiently. The main metrics to evaluate a network installation are *point-to-point bandwidth*, *latency*, and *bisection bandwidth*. However, these metrics deliver only an idealized specification (an upper bound) of the network performance without inclusion of the routing algorithm or the application communication pattern. In regular operation these values can hardly be achieved due to network congestion. The largest gap between real and idealized performance is often in bisection bandwidth which by its definition only considers the topology. The *effective bisection bandwidth* [2] is the average bandwidth for routing messages between random perfect matchings of endpoints (also known as *permutation routing*) through the network and thus considers the routing algorithm.

Routing algorithms can contain cycles in the buffer dependency graph, which might lead to global network deadlocks or generally unstable operation. It is thus most important to avoid such cycles. Several methods have been developed to guarantee deadlock-freedom, for example the controller principle [3] to manage the flows. This defines an algorithm on each node, which can permit or forbid the generation, the transfer and the consumption of packages. Another method is used by the Up*/Down* routing [4], which simply avoids cycles in the buffer dependency graph by limiting the number of possible paths during the routing process. Both methods restrict the number of possible paths through the network and might thus limit the effective bisection bandwidth. Another concept, which avoids such limitations, is the splitting of a physical channel into a set of virtual channels [5], whereby the routing can avoid a cyclic channel dependency graph through assigning paths or subpaths to different virtual channels.

A. Related Work

Deadlock-free routing strategies exist for torus topologies [5], [6] and fat-tree/Clos networks [7]. However, those schemes only guarantee high bandwidth and deadlock-freedom in case of highly structured networks. It is common practice to have special endpoints in such topologies (e.g., login nodes and file-system servers are often connected with redundant links to minimize the impact of network failures). Thus, only three of our six investigated real-world systems are pure fat-tree or torus topologies. It is also common that supercomputers are extended later and topologies grow with the machines. The properties of specialized routing

algorithms do not hold on such irregular network topologies, and deadlocks or a low effective bisection bandwidth occur in practice.

Optimal oblivious routing for general networks have been proposed but either require the solution of NP-hard problems [8] or employ linear programming techniques that are too time-consuming for large-scale systems [9], [10]. Several practically feasible algorithms have been implemented in the Open Subnet Manager of InfiniBand (OpenSM) [11]. OpenSM offers MinHop routing, which might generate cycles, the cycle-free Up*/Down* routing [12], DOR and its cycle-free variant LASH [13]. Recently, SSSP routing, a new routing algorithm that delivers higher bandwidth, has been proposed [14]. However, it is not deadlock-free on common topologies as torus.

Several approaches exist to resolve deadlocks by breaking the cycles with virtual channels in existing routing algorithms [13], [15].

In our work, we combine the fast SSSP algorithm with approaches to avoid deadlocks in general networks. We systematically study routing time, effective bisection bandwidth, and the influence on application performance on large-scale networks. The major contributions of our work are:

- We present a formal definition of the virtual channel assignment problem and a proof that finding of the minimum number of virtual channels to guarantee deadlock-freedom in general networks is NP-complete.
- We propose a high-bandwidth deadlock-free routing algorithm for arbitrary topologies and a heuristic to utilize all available virtual channels to maximize performance [16].
- We present an open-source implementation of our routing algorithm and of the virtual channel assignment, and compare effective bisection bandwidth, routing time and application performance for several artificial and real networks.

Several different network technologies are used to implement large systems: Gigabit Ethernet, InfiniBand and proprietary interconnects are widely used in modern HPC systems. Without loss of generality and due to its wide availability (41.4% of the systems in the current Top 500 list are connected with InfiniBand), we chose the InfiniBand network for our experiments and implementation. Our techniques can be used for other networks, for example Ethernet (VLAN tags can be used as virtual channels [17]) or IBM's PERCS network [18]. InfiniBand supports arbitrary network topologies and up to 16 virtual channels (called *virtual lanes*; we remark that currently available hardware only supports up to eight virtual lanes).

The paper is structured as follows: Section II presents the single-source-shortest-path routing algorithm and its benefits and issues. We define the *acyclic path partitioning* problem, which models the deadlock-prevention approach using virtual channels, in Section III. Section IV shows the extension of the SSSP routing algorithm to a deadlock-free routing (DFSSSP). In Section V we discuss the improvements of the DFSSSP routing for real-world and artificial topologies regarding the simulated effective bisection bandwidth and compare the virtual channel requirement. The last Section VI studies the performance of the DFSSSP routing on the basis of the effective bisection bandwidth and benchmarks on an existing HPC system.

II. Single-Source-Shortest-Path Routing

Single-source-shortest-path (SSSP) routing was introduced in [14]. SSSP routing globally balances the number of routes per link along all shortest paths between source and destination pairs to optimize link utilization. The network is modeled as a directed multigraph $G(V, E)$ in which V represents the set of network nodes and E represents the set of physical connections between the nodes. Balancing is done by iterative application of a single-source shortest-path algorithm, e.g., Dijkstra's algorithm [19], and by increasing all edge weights along routed paths in each iteration. The total routing algorithm, see Algorithm 1, iterates over all nodes in the graph to find the shortest paths from one source to all other nodes. The reverse path is used to generate the forwarding tables, called ft in Algorithm 1, for all nodes in order to transfer packets to the source.

The balancing of the bandwidth is reached through a weight for each edge, which will be updated in each iteration according to the following rule: Each edge weight ω_e for e will be incremented by one for all edges of the path from v to source. This will be done for all nodes $v \in V \setminus \{\text{source}\}$. Obviously, the order of the sources defines the routes and initializing the edge weights to one can cause non-minimal path lengths. As an example for the non-minimal path lengths, we show in Figure 1 the state of the graph after all routes to v_1 have been determined. Starting a new search at v_2 leads to a detour over node v_6. To force minimal paths (and thus minimal latency) we need another initialization. An appropriate initial edge weight would be $|V|^2$, because a detour to avoid edge e with weight $|V|^2 + \omega_e^i$ needs at least two edges e' and e''. Here ω_e^i means the additional weight after the $i.$ iteration, and $\omega_e^i < |V|^2$ through the fact that only $|V| \cdot (|V| - 1)$ paths are observed. From this it fol-

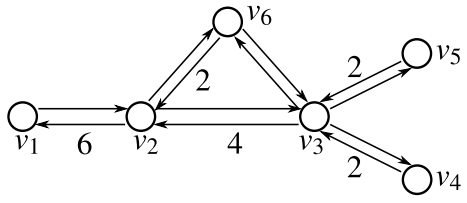

Figure 1. Updated weights after first iteration with source v_1

Algorithm 1 SSSP Routing

Input: Graph(V,E)
Output: Forwarding Tables

```
for all nodes w ∈ V do
  source ← w
  /* search shortest path from source to all targets */
  for all v ∈ V do
    initialize v (v.distance ← ∞, v.parent ← ∅)
  end for
  source.distance ← 0
  Q ← V
  while Q ≠ ∅ do
    search u ∈ Q with u.distance ≤ v'.distance ∀ v' ∈ Q
    Q ← Q \ {u}
    for all (u, v) = e ∈ E with v ∈ Q do
      if u.distance + ω_e < v.distance then
        v.distance ← u.distance + ω_e
        v.parent ← e
      end if
    end for
  end while
  /* update edge weights */
  for all e ∈ E do
    count paths with e ∈ path(v, source) ∀ v ∈ V
    ω_e ← ω_e + #paths
  end for
  /* update forwarding tables */
  for all v ∈ V do
    if v.parent ≠ ∅ then
      v.ft[source] ← v.parent
    end if
  end for
end for
```

lows that $|V|^2 + \omega_e^i < (|V|^2 + \omega_{e'}^i) + (|V|^2 + \omega_{e''}^i)$. So our shortest-path algorithm never chooses a detour. A detailed explanation of the algorithm can be found in [14].

The SSSP routing algorithm supports arbitrary network topologies and network technologies with either source routing or distributed routing. However, as discussed in [14], SSSP routing might introduce cyclic dependencies between network buffers which might lead to network deadlocks. We will discuss such deadlock situations and possible solutions in the following section.

III. Deadlocks in General Networks

A deadlock situation can occur in a system if the following four conditions are met [20]:

1) Tasks have exclusive access to the resources;
2) A task does not release resources while waiting for additional resources;
3) Only the task, which holds a resource, can release this resource;
4) A circular dependency in which each task requests a resource held by another task in this cycle.

These conditions are sufficient for a deadlock, iff there exists only one resource. For multiple resources of the same type the conditions 1) – 4) are only necessary for a deadlock in a network.

Now, we show that the SSSP algorithm might lead to network deadlocks due to limited buffering in the switching elements. Considering, for example, a ring topology, see Figure 2, with five nodes and at most one bi-directional link between two nodes, and assuming a communication pattern in which each node sends messages to a node that is two hops away in clockwise direction, the above-mentioned SSSP routing strategy would route all messages in clockwise direction. Each message needs buffer space at the next hop to be forwarded. With the described configuration, all buffers could fill up and no message would be able to progress. This circular buffer dependency can thus lead to a deadlocked configuration.

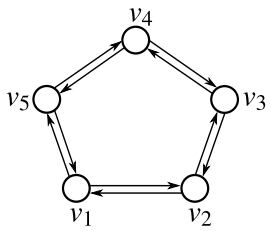

Figure 2. Ring topology that leads to deadlocked configuration with SSSP routing

Dally and Seitz defined a method for deadlock-free oblivious routing based on virtual channels [5]. They model an *interconnection network* $I := G(N, C)$ as a directed graph that consists of the node set N of processing nodes and the edge set C of communication channels in the network. The *routing function* $R : C \times N \rightarrow C$ assigns for each channel-destination pair (c_i, n_d) the next channel c_{i+1} in the path. So the path of a message through the network is only defined by the current channel and its destination. These two definitions will be used to define the *channel dependency graph* whereby one can specify whether a routing function R is deadlock-free or not. The *channel dependency graph* $D := G(C, E)$ is a directed graph with the edge set of I as nodes. The edges $e := (c_i, cj)$ of D are defined by the routing function R through $e \in E \Longleftrightarrow \exists n \in N : R(c_i, n) = c_j$. Theorem 1 in [5] states that a routing function is deadlock-free iff the corresponding channel dependency graph is acyclic. Cycles in the dependency graph can now be broken by assigning parts of the paths to different virtual channels (separate buffers). Lysne et al. define a similar method which uses virtual channels to define virtual layers and assign paths from source to destination to one layer so that each layer is deadlock-free [13].

For the sake of completeness, we note that the "iff" statement in Dally's Theorem 1 is not fully correct. Schwiebert presents a counterexample in [21]. Under certain conditions a channel dependency graph can have a cycle but the network with a defined routing function, which induced the cycle, can never reach a deadlock state. This is called an *unreachable configuration*. From this it follows that a cycle-free channel dependency graph is only a sufficient condition for a deadlock-free routing, but is no longer necessary. We use this sufficient condition in Section IV to extend the SSSP routing to a deadlock-free single-source-shortest-path routing function.

A. The Acyclic Path Partitioning Problem

Each virtual channel occupies physical resources (buffer space) in all switches. Thus, the number of such channels is often limited by a small constant, e.g., the InfiniBand specification defines a maximum number of sixteen, but all current implementations only support eight virtual lanes. Thus, the very important question, *how many virtual layers are needed for a given network and routing function to avoid deadlocks*, needs to be discussed.

To answer this question, we have to make some definitions: For a given network I and routing function R the corresponding channel dependency graph $D = G(C, E)$ is the same, as defined above. The route of a message from source to destination characterizes a path $p := c_0 c_1 \dots c_n$, $c_i \neq c_j$ for $i \neq j$, in D. Additionally, the node and edge set of a path is defined through

$$\begin{aligned} nodes(p) &:= \{\, c \in C \mid \exists\, c_i \in p : c = c_i \,\} \\ edges(p) &:= \{\, e \in E \mid \exists\, c_i, c_{i+1} \in p : e = (c_i, c_{i+1}) \,\} \end{aligned}$$

A set P of paths is a *generator* of D, if the conditions

$$V_P := \bigcup_{p \in P} nodes(p) \quad \text{and} \quad E_P := \bigcup_{p \in P} edges(p)$$

hold for the induced graph $D \cong G[P] := (V_P, E_P)$. A *partition* $\mathcal{P} := \{\, P_1, \dots, P_k \mid \forall\, i = 1, \dots, k : P_i \subseteq P \,\}$ of the generator P is called *cover* iff

i. $P_i \neq \emptyset$ for all $i = 1, \dots, k$
ii. $P = \bigcup_{i=1}^{k} P_i$
iii. $\forall\, i, j = 1, \dots, k,\ i \neq j : P_i \cap P_j = \emptyset$ and
iv. the generated graph $G[P_i]$ is acyclic for all $i = 1, \dots, k$.

These definitions will be used to postulate the following decision problem

Instance:	Given is the generator P and a positive integer $k < \|P\|$.
Question:	Is there a partition $\mathcal{P} = \{P_1, \dots, P_k\}$ of P so that $\mathcal{P}$ is a cover?

which we call the *acyclic path partitioning* problem or APP problem. We show an example of this problem in Figure 3. It shows a generator $P = \{p_1, p_2, p_3\}$, with $p_1 = bc$, $p_2 = abc$ and $p_3 = cdab$ on the left, and the possible cover $\mathcal{P} = \{P_1 = \{p_1, p_{p2}\}, P_2 = \{p_3\}\}$ on the right side.

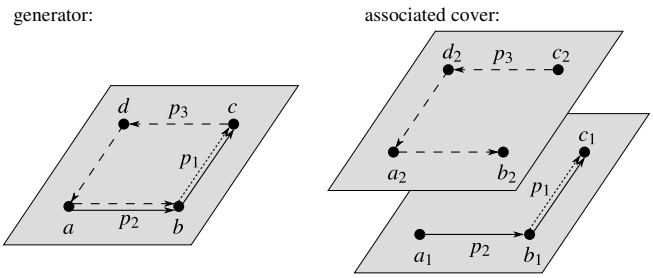

Figure 3. Example for $P = \{p_1, p_2, p_3\}$ and $k = 2$

Next, we will show that no efficient algorithm is known to solve the APP problem.

Theorem 1. *The acyclic path partitioning problem is NP-complete.*

Proof: A decision problem Π is *NP-complete* [22] iff

i. $\Pi \in$ NP and
ii. Π is NP-hard.

The proof of i. is trivial, because a non-deterministic algorithm can guess a truth assignment $g : P \to \mathbb{N}$ for each path $p_i \in P$, so that p_i is part of one subset P_j. The algorithm can validate the partitioning in polynomial time as follows: It searches in each generated graph $G[P_j]$, for all $1 \leq j \leq k$, for cycles via a depth-first search. Deduced from the time complexity $\mathcal{O}(|V| + |E|)$ for the DFS on a graph $G(V, E)$ follows the polynomial time for the validation. The proof of ii. is more complicated. A problem Π is *NP-hard* iff

$$\forall\, \Pi' \in \text{NP} : \Pi' \leq_p \Pi$$

which means for each problem in NP there is a polynomial transformation into the problem Π. With the assistance of Lemma 2.1 and Lemma 2.2 [22] it is sufficient to show a polynomial transformation from one arbitrary problem $\Pi' \in$ NP into the problem Π. For sake of completeness a function f is called *polynomial transformation* iff

(a) the computation of f can be done in polynomial time with a deterministic Turing machine and
(b) for all instances x of Π' with $x \in \Pi' \iff f(x) \in \Pi$.

Now a polynomial transformation from the graph k-colorability problem (GT4 [22]) for the graph $G(V, E)$ into the APP problem will be constructed. For $k \geq 3$ GT4 is NP-complete. First the set of adjacent nodes $adj(v) := \{\, w_1, \dots, w_m \mid \forall\, 1 \leq i \leq m : w_i \in V \wedge (v, w_i) \in E \,\}$ is defined to construct the paths in P. For each node $v \in V$

two set will be defined

$$
\begin{aligned}
V_v &:= \{\langle v\rangle\} \cup \{\, \langle v, \{v,w\}\rangle \mid w \in adj(v) \,\} \\
&\qquad \cup \{\, \langle w, \{v,w\}\rangle \mid w \in adj(v) \,\} \\
E_v &:= \{\, (\langle v\rangle, \langle v, \{v,w_1\}\rangle) \,\} \\
&\quad \cup \{\, (\langle v, \{v,w_i\}\rangle, \langle w_i, \{v,w_i\}\rangle) \mid \\
&\qquad\qquad w_i \in adj(v)\ \forall\, 1 \le i \le m \,\} \\
&\quad \cup \{\, (\langle w_i, \{v,w_i\}\rangle, \langle v, \{v,w_{i+1}\}\rangle) \mid \\
&\qquad\qquad w_i, w_{i+1} \in adj(v)\ \forall\, 1 \le i < m \,\}
\end{aligned}
$$

where the set E_v consists of directed edges. From this it follows that for all $v \in V$ the graph $G(V_v, E_v) =: p_v$ only consists of an acyclic directed path with

- $p_v = \langle v\rangle \iff adj(v) = \emptyset$
- $p_v = \langle v\rangle\langle v, \{v,w_1\}\rangle\langle w_1, \{v,w_1\}\rangle \ldots \langle v, \{v,w_m\}\rangle$ $\iff |adj(v)| = m$

Hence, the generator P can be constructed from V and E in polynomial time by defining

$$P := \{\, p_v \mid v \in V \,\}$$

whereby (a) has been proved. Based on the definitions above, two proposition can be made

(1) Let $(v,w) \in E \implies$ the graph $G[\{p_v, p_w\}] = G(V_v \cup V_w, E_v \cup E_w)$ is cyclic, because

$$
\begin{aligned}
p_v &= \langle v\rangle \ldots \langle v, \{v,w\}\rangle\langle w, \{v,w\}\rangle \ldots \\
p_w &= \langle w\rangle \ldots \langle w, \{w,v\}\rangle\langle v, \{w,v\}\rangle \ldots
\end{aligned}
$$

create the cycle $\langle v, \{v,w\}\rangle\langle w, \underbrace{\{v,w\}}_{=\{w,v\}}\rangle\langle v, \underbrace{\{w,v\}}_{=\{v,w\}}\rangle$.

(2) Let $V' \subseteq V$ be an independent set $\implies$ $\bigcup_{v \in V'} G(V_v, E_v) =: G_{V'}$ is acyclic. This follows from $\forall\, v,w \in V' : V_v \cap V_w = \emptyset \implies$ no edge exists between p_v und $p_w \implies G_{V'}$ consists of disjoint acyclic paths.

Finally, a proof is needed which verifies that the construction of the generator P is a polynomial transformation. Referring to (b) first the proof for "⇒": Let $\{V_1, \ldots, V_k\}$ be the graph k-coloring of $G(V,E)$ where each node in V_i has the color i $\overset{\text{def.}}{\implies}$ the sets V_i are each independent sets $\overset{(2)}{\implies}$ $\{P_1, \ldots, P_k\}$ is a k-cover, whereby $P_i := \{\, p_v \mid v \in V_i \,\}$ for $1 \le i \le k$. For the opposite direction "⇐": Let $\{P_1, \ldots, P_k\}$ be a k-cover $\overset{\text{def.}}{\implies}$ $G[P_i]$ is acyclic for all $1 \le i \le k$ $\overset{(1)}{\implies}$ for any two $v,w \in V$, with $p_v, p_w \in P_i$, there is no edge (v,w) in $E \implies$ the sets V_i, which are the base for P_i, are independent sets $\implies$ $\{V_1, \ldots, V_k\}$ is a k-coloring. ■

In the following section, we present an efficient heuristic for the *acyclic path partitioning* that minimizes the number of virtual layers in deadlock-free SSSP routing.

IV. Deadlock-Free SSSP Routing

Our first approach is similar to Layered Shortest Path Routing (LASH) [13]. It starts with the generation of the shortest path between each source-destination pair in the network (see Algorithm 1). After this step, the online algorithm searches a layer for each path, so that adding path x to layer y does not close a cycle in the channel dependency graph of layer y. This results in one cycle search per path, at the minimum. This approach is very time-consuming and thus not scalable to large networks. The depth-first search on the channel dependency graph $D = G(C,E)$ has a complexity of $\mathcal{O}(|C|+|E|)$. The channel dependency graph grows successively when the paths are added. For a network $I = G(N,C)$ there are $|N|^2$ paths, so the minimum time complexity for the online algorithm is $\mathcal{O}(|N|^2 \cdot (|C|+|E|))$. This is impractical for large-scale networks.

Our second approach first creates all paths and then breaks cycles in the complete graph D in an offline-manner (see Algorithm 2). The initial channel dependency graph contains all paths and possibly cycles. Then, a cycle search is started for this layer, which aborts when a cycle is found and returns the cycle. The algorithm cuts the cycle by taking one edge of the cycle and moving all paths, which induce this edge, to the next layer. Next, the cycle search is resumed on the same place, where the search aborted initially. This is done until no cycles remain in the first layer. Relocated paths can create new cycles in the channel dependency graph of the next layer. Hence, all layers have to be processed in the same manner until each channel dependency graph is acyclic.

The advantage of the offline algorithm is, that exactly one complete cycle search for each channel dependency graph is needed. This reduces the duration enormously, especially for large network configurations, but at the cost of a higher memory complexity. For example, a synthetic network with 4096 nodes was processed in approx. 170 seconds by the offline algorithm instead of nearly two hours with the online method, while the additional memory requirement of the offline algorithm was 340 MByte. The higher memory complexity results from the storage of all paths which induce an edge in the channel dependency graph. Thus, an attribute of each edge is an one-dimensional list of source-destination pairs, whose path induces the edge. This list is needed by the offline algorithm to identify the paths which have to be moved to the next layer to break a cycle. Let $d(I)$ be the graph diameter of the network I. So the longest path consists of $d(I)-1$ sequential channels. From this it follows that the memory complexity of one channel dependency graph with $|N|^2$ paths is $\mathcal{O}(d(I) \cdot |N|^2 + x)$ for the offline algorithm, where $\mathcal{O}(x) = \mathcal{O}(|C| + |E|)$ is the memory complexity of the same channel dependency graph for the online algorithm. The highest additional memory requirement of 540 MByte, we measured in our simulations, was for the Ranger cluster, which was mentioned in the introduction.

Algorithm 2 Search and Remove Deadlocks

Input: Graph(V,E), Forwarding Tables
Output: Path to Virtual Layer Assignment

```
/* initialize first channel dependency graph */
for all nodes u, v ∈ V do
    update cdg[1] with path(u, v)
    assign the virtual layer for this path u.ft[v].vl ← 1
end for
/* search for cycles in the channel dependency graphs */
for i = 1 to #{virtual layers} − 1 do
    repeat
        search for a cycle in cdg[i]
        identify weakest edge of the cycle
        for all path(u, v) induces the weakest edge do
            remove path(u, v) from cdg[i]
            update cdg[i + 1] with path(u, v)
            u.ft[v].vl ← i + 1
        end for
    until no cycle found in cdg[i]
end for
search for a cycle in cdg[#{virtual layers}]
if cycle found then
    no deadlock-free assignment possible
end if
/* balance paths on empty CDGs without additional
   cycle search */
c ← #{non-empty CDGs}
for i = 1 to c do
    calculate a set S of empty virtual layers for cdg[i]
    for all j ∈ S do
        /* move paths from cdg[i] to cdg[j] */
    end for
end for
```

We summarize our findings in two propositions. First, we look at the parts of Algorithm 1 and 2 step by step. Dijkstra's algorithm has a time complexity of $O(|N| \cdot \log |N| + |C|)$ using a binary or Fibonacci heap to search the node with smallest distance and a memory complexity of $O(|N|+|C|)$. The time complexity to create a channel dependency graph is $O(|N|^2)$ while it needs $O(|C| + |E|)$ memory. The cycle search, as indicated ahead, has a time complexity of $O(|C| + |E|)$. Additionally, we define the minimum number of virtual layers

$$\nabla := \min\{\, k \in \mathbb{N} \mid \exists\, k\text{-cover for } I \text{ and } R \,\}$$

for the following propositions.

Proposition 1. *The time complexity of the online DFSSSP algorithm is*

$$O(\, |N|^2 \cdot (\nabla \cdot (|C| + |E|) + \log |N| + 1) + |N| \cdot |C| \,)$$

while its memory complexity is

$$O(\, \nabla \cdot (|C| + |E|) + |N| \,)$$

Proposition 2. *The time complexity of the offline DFSSSP algorithm is*

$$O(\, |N|^2 \cdot (\log |N| + \nabla) + |N| \cdot |C| + \nabla \cdot (|C| + |E|) \,)$$

while its memory complexity is

$$O(\, \nabla \cdot d(I) \cdot |N|^2 + \nabla \cdot (|C| + |E|) + |N| \,)$$

The last question which remains is, *which edge of the cycle we have to break to minimize the number of virtual channels needed for a deadlock-free routing.* With reference to Section III-A, no efficient algorithm is known to minimize the number of virtual channels. We implemented the simple heuristic to remove the weakest edge in the cycle. This means to remove the edge induced by the least number of paths to minimize the number of paths in the next virtual channel. The second heuristic does the opposite. It removes the edge induced by the largest number of paths to break currently undiscovered cycles induced by the same paths. Additionally, we tested a pseudo-random heuristic which removes all paths of the first discovered edge of a cycle.

Several other heuristics are known to optimize NP-complete problems like APP, e.g., *random optimization*, *genetic algorithms*, *simulated annealing* or *threshold accepting*. However, the application of those methods suffers from two problems: The first one is the absence of a weighting function for the APP problem. Only two possible states are known: deadlocked configuration and deadlock-freedom. Hence, heuristics like *simulated annealing* cannot be adapted to the APP problem. The next problem is the time complexity, algorithms like *random optimization* would cause more validation steps (i.e., time-consuming cycle searches), whether the partitioning is deadlock-free or not.

Simulations with random topologies, consisting of 64 switches, 1024 endpoints, and 128 links to connect the switches, revealed that the best of our heuristics is to break the weakest edge of the cycle. Depending on the random topology, the needed number of virtual channels ranges from 3 to 5 for this heuristic, whereas the range for the pseudo-random heuristic is 4 to 8. The third was the worst heuristic with a range from 4 to 16.

V. Simulated Bandwidth and Virtual Channels

To evaluate the quality of the developed routing methods, we used the Oblivious Routing Congestion Simulator (ORCS, [23]). The simulator calculates the effective bisection bandwidth from an given arbitrary network topology and routing. ORCS reads a directed graph representation of the network, which also includes the routing information. It generates a user-defined number of random bisection patterns, whereby one endpoint in subset A has exactly one associate endpoint in subset B (random perfect matching).

Thus, the simulator assumes one communication process per node. ORCS simulates a point-to-point communication among all pairs and counts the congestion to evaluate the relative effective bisection bandwidth of the network.

We compared our proposed routing algorithms with the routing algorithms supplied by the InfiniBand Subnet Manager, MinHop, Up*/Down*, FatTree, LASH and Dimension Order Routing, for six real-world HPC systems. The real-world examples are the 550-node CHiC at Chemnitz University of Technology, the 3288-node system JUROPA/HPC-FF at Forschungszentrum Jülich, the 128-node system Odin operated by Indiana University and the Ranger HPC system, which consists of 3936 nodes at Texas Advanced Computing Center. Additionally, we have the graph representations of a 1430-node configuration of the Tsubame HPC system at Tokyo Institute of Technology and the 724-node Deimos cluster, which is described in detail in Section VI. All systems use an InfiniBand interconnect. The results are displayed in Figure 4, a missing bar indicates that the selected routing algorithm failed. SSSP and DFSSSP routing has a significantly higher effective bisection bandwidth than the other algorithms. The only exception is the Odin cluster, which is a pure fat tree with only one 144-port switch. Besides the 4.75% regression in the simulated effective bisections bandwidth for Odin, the distance between the DFSSSP routing and second best algorithm are in the range of 1.4% for JUROPA to 63.27% improvement for Ranger.

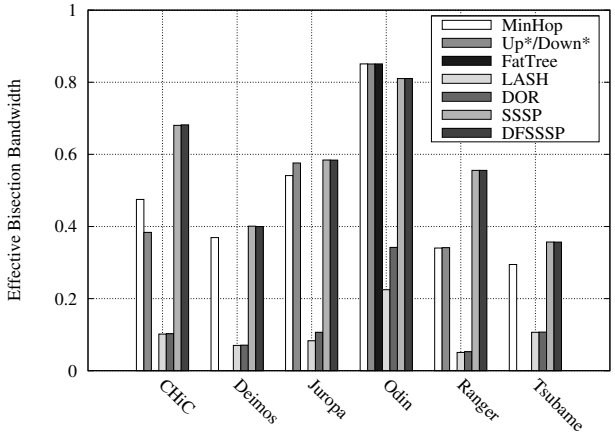

Figure 4. Simulated bandwidth for real-world HPC systems; 1000 random bisection pattern

On the other hand, we investigated some artificial network topologies commonly used for HPC systems. Figure 5 shows the results for extended generalized fat trees (XGFT) [24]. On one side, the bandwidth of the LASH and DOR routing is decreasing steadily. On the other side, the effective bisection bandwidth of the routing algorithms MinHop, Up*/Down* and DF-/SSSP is relatively constant for one height h of the tree. For the height $h = 2$ of the tree, i.e., starting from

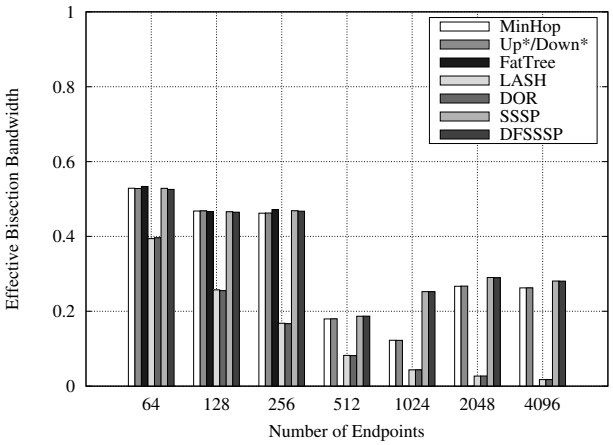

Figure 5. Simulated bandwidth for a XGFT network; 1000 random bisection pattern

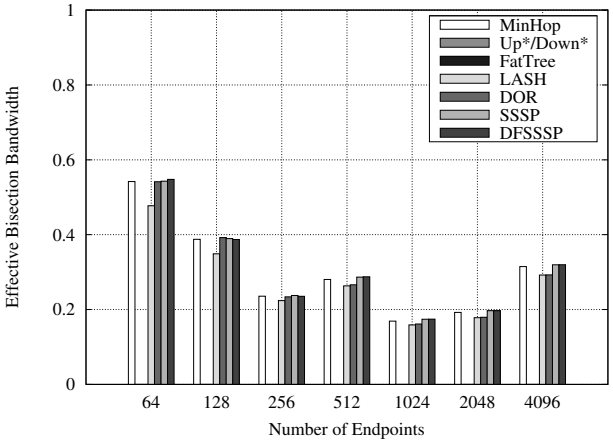

Figure 6. ORCS: effective bisection bandwidth simulations for a Kautz network; 1000 random bisection patterns

512 endpoints, there is a small increase in bandwidth of DF-/SSSP compared to the MinHop routing. One special case is the configuration with 1024 endpoints. The effective bisection bandwidth of the DF-/SSSP routing is approx. twice as high as the value of MinHop. From these facts it follows that the load balancing, and in this way the congestion minimization, of our developed routing algorithms is superior for those topologies.

Another investigated topology are networks designed as Kautz graphs [25], whereby the switches build the Kautz graph and endpoints are connected to them. The advantage of Kautz graphs is the minimal graph diameter ensuring short paths in the network. As it is shown in Figure 6, all investigated routing algorithms provide similar effective bisection bandwidths for this type of topology. Each increase in bandwidth, during an increment in the number of endpoints, is associated with an increase in the number of links connecting the switches. These connection links are

proportional to the parameter b, see Table I. In contrast to Figure 5, LASH provides nearly the same bandwidth as the deadlock-free version of the SSSP routing.

Additionally to the bandwidth simulations, we measured the runtime for each routing algorithm on a common workstation. An example of these measurements is shown in Figure 7 and we can deduce that our offline DFSSSP algorithm has $\approx$ 10x higher runtime with respect to the MinHop algorithm. The topology for this measurement is a k-ary n-tree [26]. This fact holds for the real-world HPC systems, too, as one can see in Figure 8.

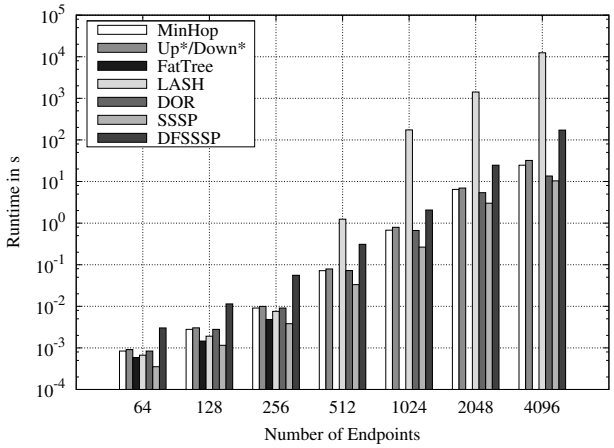

Figure 7. Runtime comparison for k-ary n-tree topologies

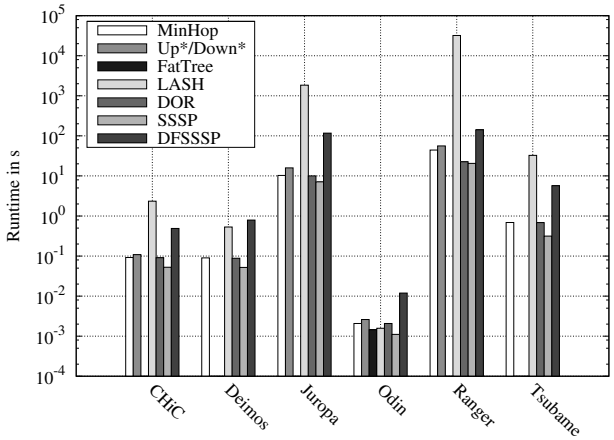

Figure 8. Runtime comparison for real-world HPC systems

The parameters to create the different network sizes (number of endpoints) for the XGFT, Kautz and k-ary n-tree topology are presented in the following Table I. We assumed the use of 36-port switches.

A. Needed Number of Virtual Layers

Furthermore, we investigated the number of virtual layers needed for a deadlock-free routing. The reference value

	Switch Topologies		
#Endpoints	XGFT(h; m; w)	Kautz(b, n)	k-ary n-tree
64	XGFT(1; 6; 3)	Kautz(2, 2)	6-ary 2-tree
128	XGFT(1; 10; 5)	Kautz(2, 2)	10-ary 2-tree
256	XGFT(1; 16; 8)	Kautz(2, 3)	16-ary 2-tree
512	XGFT(2; 6,6; 3,3)	Kautz(3, 3)	6-ary 3-tree
1024	XGFT(2; 10,10; 5,5)	Kautz(3, 3)	10-ary 3-tree
2048	XGFT(2; 14,14; 7,7)	Kautz(4, 3)	14-ary 3-tree
4096	XGFT(2; 18,18; 9,9)	Kautz(6, 3)	18-ary 3-tree

Table I
PARAMETER TO GENERATE THE NETWORKS

for this measurement will be the LASH algorithm. It is important to achieve a small number of virtual channels, because they might be bounded as described in Section I. With respect to [15], we first measured the number of virtual layers for random topologies. In Figure 9, we present the minimum, maximum and average number of virtual layers needed for a random topology. We use a network with 128 32-port switches, each connected to 16 endpoints, random connections between the switches, and we vary the number of such connections. For each number of connections, 100 random topologies are generated and the number of needed virtual layers for the LASH and DFSSSP routing are computed by applying the routing algorithm. The analysis shows, that the average number of virtual layers for LASH is smaller for a larger number of connection links (a denser network),

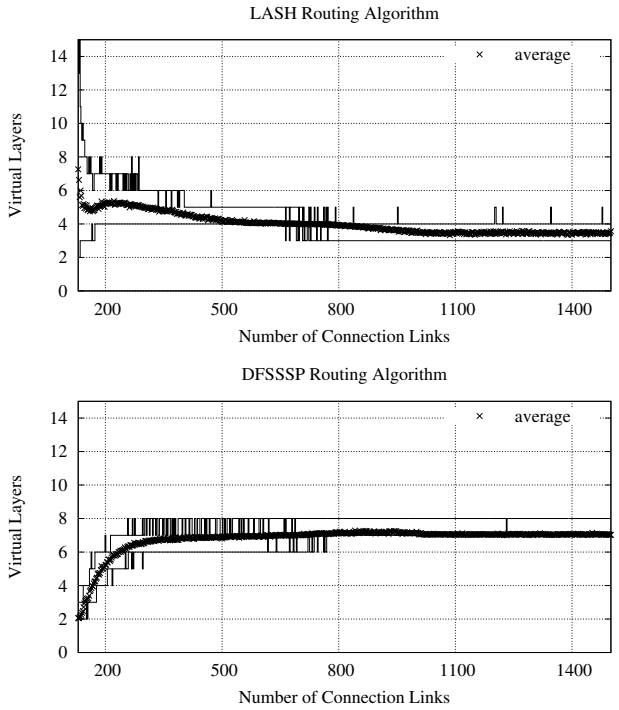

Figure 9. Number of needed virtual layers for a deadlock-free routing of random topologies

but the DFSSSP routing performs better for a lower number of connection links (a sparser network). The intersection point for the average number is at about 200 connections.

The more relevant question is, how many virtual layers are needed to route real-world HPC systems? Figure 10 shows the required number of virtual layers for each topology. DFSSSP routing performs better on these topologies. However, due to the NP-complete decision problem, it remains unclear whether the LASH routing algorithm or the DFSSSP algorithm performs better in general.

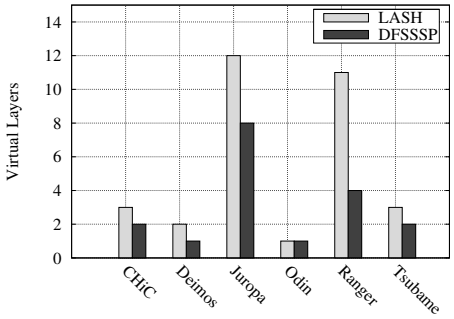

Figure 10. Virtual layers needed for real-world HPC systems

VI. Experimental Results

To validate our simulated results we implemented the SSSP routing and the DFSSSP routing as part of the InfiniBand Subnet Manager and used a real-world cluster to measure the performance benefits based on synthetic and application benchmarks. The cluster, named Deimos, consists of 724 nodes with 2576 cores and is operated by the Center for Information Services and High Performance Computing at Technische Universität Dresden. The interconnect network of Deimos is composed of three 288-port fat-tree switches, connected with 30 links, see Figure 11. The remaining ports of the switches are used to connect the endpoints, which are equipped with PCIe 1.1 HCAs to deliver a theoretical peak bandwidth of 946 MiB/s for a point-to-point connection.

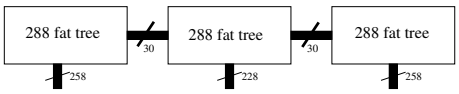

Figure 11. Deimos network topology

The reference for our measurements will be the MinHop routing algorithm, which is part of the InfiniBand Subnet Manager and delivers the second-highest bandwidths [14] besides the SSSP/DFSSSP algorithm but is not deadlock-free. Additionally, the figures will contain the results of the LASH routing algorithm, because it uses the same approach to be deadlock-free but we remark that LASH was designed for torus topologies and we do not expect high bandwidths on Deimos' topology. All measurements on Deimos were done in the same way. For the number of cores up to 512, we used one core per node. For the measurements with 1024 cores the MPI processes were spread across 250 nodes using a mixture of dual-, quad- and octa-core nodes. We used the same nodes (allocation) for identical number of cores so that the only difference was the routing we used.

A. Measured Bandwidths and Microbenchmarks

Netgauge [27] is a network measurement tool which supports benchmarking of many different network protocols and communication patterns. One part of Netgauge is the determination of the effective bisection bandwidth. This benchmark partitions the processes into two equal sets, A and B. Each process of set A communicates with exactly one process of set B. Several random partitions are investigated to measure the average bandwidth. We used this tool to verify the simulated results of Section V for Deimos. After the synchronization step all pairs of processes perform a ping-pong communication of 1 MiB for 50 iterations. We set the number of random partitionings to 1000 to get an adequate approximation of the effective bisection bandwidth. Figure 12 shows the results of our measurements. Thereby, the improvement of our deadlock-free routing algorithm grows from 27% for 128 cores over 47% to a doubling of the bandwidth for 512 cores. But the absolute effective bisection bandwidth decreases for all routing algorithms because of an increase of the congestion in the interconnect network of Deimos. The absolute gap between the MinHop and DFSSSP routing corresponds to 69.7 MiB/s – 79.3 MiB/s per pair for the first three steps.

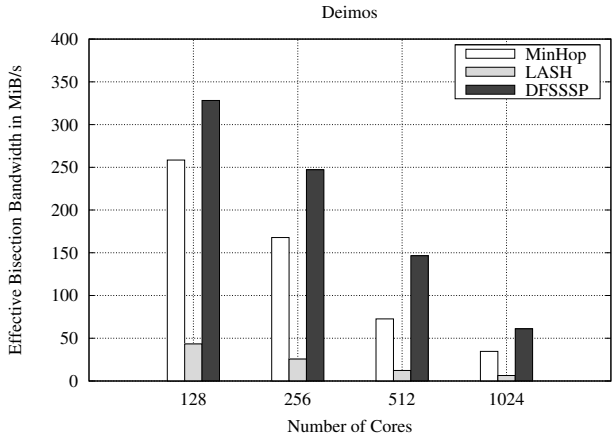

Figure 12. Netgauge: effective bisection bandwidth measurements

In addition to the Netgauge benchmark we used some communication microbenchmarks of BenchIT [28]. BenchIT is a measuring environment to provide a set of parallel and non-parallel kernels for a wide range of problems. It is able to manage the measurements and can display and compare

the results. It was developed to compare architectures by using fixed microbenchmarks or to compare different implementations of the same algorithm, e.g., all loop permutations for a matrix×matrix multiplication algorithm.

First, as a reference measurement, we measured the point-to-point bandwidth of a synchronous send. The kernel operates as follows: Rank 0 of the MPI processes sends a fixed number of bytes to all other processes and measures the minimal, maximal and average bandwidth. We ran this microbenchmark on 128 cores, one core per node, and the results were as expected, all routing algorithms delivered the same bandwidths due to the absence of congestions and shortest path routing. For this measument, with message sizes up to 2.5 MiB, all routings achieved an average bandwidth of 844.65 MiB/s at the largest message size. To investigate congested situations, we ran a microbenchmark that measures the runtime of collective communications. The result is shown in Figure 13. The kernel iterates over the number of floats in the send buffer and measures the time for the MPI all-to-all operation. For the largest number of 4096 floats the processes transfer accumulated 254 MiB. As one can see, and as the Netgauge benchmarks has indicated, the DFSSSP routing algorithm balances the network traffic and reduces the congestions better than the MinHop algorithm. So our routing algorithm leads to a processing time of 10.06 ms instead of 18.88 ms in the case of 4096 floats in the send buffers, which corresponds to a speedup of 46.7%.

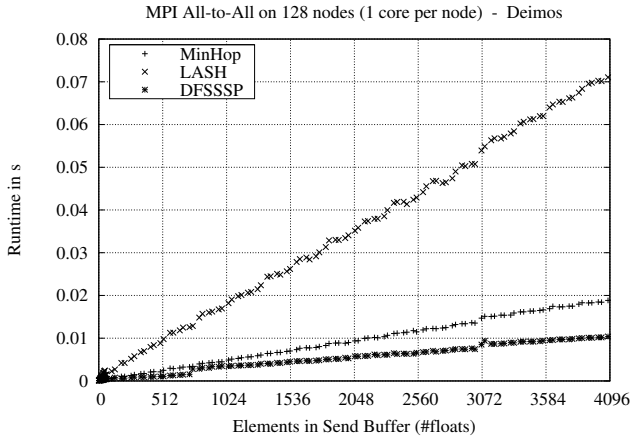

Figure 13. BenchIT: MPI all-to-all communication microbenchmark

B. Application Benchmarks

For the application benchmarks we used the original MPI-based NAS Parallel Benchmarks suite [29] version 2.4. It provides, apart from an embarrassingly parallel benchmark and an integer sort kernel, six parallel benchmarks. We will mainly concentrate on the results of the benchmarks BT, FT and SP and present the improvements of the DFSSSP routing in tabular form for 1024 cores, see Table II.

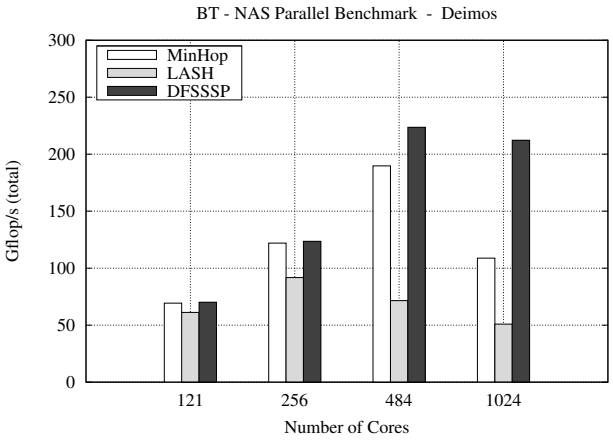

Figure 14. BT solves a system of equations as used in CFD codes

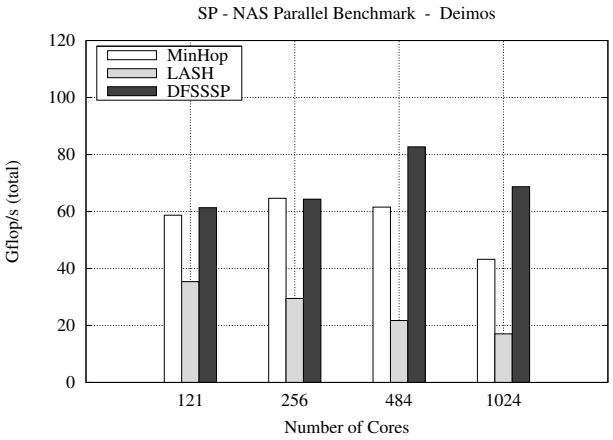

Figure 15. SP solves a system of equations as used in CFD codes

Both the BT and SP benchmark solve a system of equations whereby, in the first case, the matrix has a block tridiagonal structure and in the second case a scalar pentadiagonal structure, so the solvers have a different computation-to-communication ratio. The nearest-neighbor communication in these codes is done over non-blocking point-to-point communications and, additionally, a small amount of all-reduce operations are processed. As one can see in Figure 14 and Figure 15 the MinHop routing algorithm performs as well as our DFSSSP algorithm for the smaller number of cores, 121 and 256. The small number of cores and nearest-neighbor communication does not cause to much congestion in the interconnect network. For 484 cores the performance values of the BT benchmark diverge, but both routing algorithms lead to a positive scaling of the code. Whereas the SP code has a drop in the performance for this number of cores for the MinHop routing. But the DFSSSP

routing induces a performance gain and the communication overhead will be crucial for 1024 cores, for the first time.

The FT kernel solves a 3-dimensional partial differential equation applying fast Fourier transformations. The difference in communication, with respect to the BT and SP kernel, is that the FT benchmark uses MPI collective operations, mainly all-to-all communication and some reduce operations. So when communication is performed, it involves all processes at the same time and our measurements should show an improvement of the DFSSSP routing even for smaller numbers of cores. We validate these assumptions in our measurements, shown in Figure 16. Even for 128 and 256 cores the DFSSSP routing delivers an improvement of $\approx 25\%$ for the Gflop/s rate.

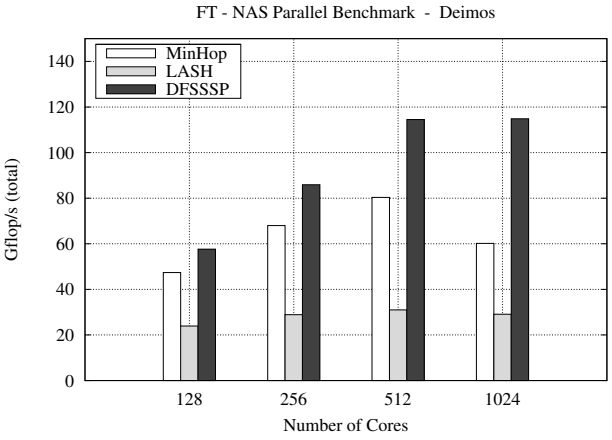

Figure 16. FT solves a 3D FFT partial differential equation

	Gflop/s (total)		
Benchmarks	MinHop	DFSSSP	Improvement in %
BT	108.82	212.27	95.07
CG	5.81	8.75	50.60
FT	60.16	114.87	90.94
LU	129.26	247.46	91.44
MG	164.84	215.51	30.74
SP	43.21	68.71	59.01

Table II
NAS PARALLEL BENCHMARKS FOR 1024 CORES ON DEIMOS

The kernels CG, LU and MG showed similar performance characteristics and thus are omitted. Table II shows the performance improvements for all measured application benchmarks for 1024 cores which are in the range of 30% to 95%.

VII. SUMMARY AND CONCLUSION

We demonstrated that the SSSP routing algorithm can be affected by network deadlocks. It is possible to use virtual channels (layers) to break cycles in the channel dependency graph, however, optimal assignment of virtual channels to routes is unclear. We formally defined the acyclic path partitioning (APP) problem which models the cycle-free assignment of routes to virtual layers. We proved that the APP problem is NP-complete. We proposed three different heuristics to solve the problem out of which one, breaking the weakest edge of the cycle, is the most suitable for a practical implementation. Our implementation of DFSSSP (deadlock-free SSSP) in OpenSM uses this heuristic to create a deadlock-free routing for arbitrary network topologies.

We demonstrated that DFSSSP routing enables higher bandwidth than comparable algorithms (e.g., MinHop which is not deadlock-free) and uses less virtual layers than LASH on existing cluster systems. We also investigated DFSSSP on a 724-node cluster system and benchmarked a doubling in effective bisection bandwidth and an improvement for application performance of up to 95%.

Our implementation is ready to be used in InfiniBand production environments and improves network performance transparently. A patched version of the OpenSM is available at `http://unixer.de/research/dfsssp/`. Although our implementation is InfiniBand-specific, the algorithms apply to generic networks.

Acknowledgments

The authors thank Christel Baier (TU Dresden) for the idea and an initial outline of the proof for Theorem 1.

We also thank Guido Juckeland (TU Dresden) for support during the experiments with the Deimos cluster. Thanks to Bernd Mohr (FZ Juelich) who provided the JUROPA topology, Hideyuki Jitsumoto and Satoshi Matsuoka who provided the Tsubame topology, and Len Wisniewski (Sun) and the TACC who provided the Ranger topology.

REFERENCES

[1] P. Kogge, K. Bergman, and S. Borkar, "Exascale computing study: Technology challenges in achieving exascale systems," University of Notre Dame, Department of Computer Science and Engineering, Notre Dame, Indiana, Tech. Rep. TR-2008-13, Sep. 2008.

[2] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *Proceedings of the IEEE International Conference on Cluster Computing (10th CLUSTER'08)*. Tsukuba, Japan: IEEE, Sep.-Oct. 2008, pp. 116–125.

[3] S. Toueg, "Deadlock- and livelock-free packet switching networks," in *STOC '80: Proceedings of the twelfth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1980, pp. 94–99.

[4] J. C. Sancho, A. Robles, and J. Duato, "A flexible routing scheme for networks of workstations," in *ISHPC '00: Proceedings of the Third International Symposium on High Performance Computing*. London, UK: Springer-Verlag, 2000, pp. 260–267.

[5] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, 1987.

[6] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1981, pp. 263–277.

[7] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem, "Oblivious routing for fat-tree based system area networks with uncertain traffic demands," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 337–348, 2007.

[8] H. Räcke, "Minimizing congestion in general networks," in *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 43–52.

[9] M. Bienkowski, M. Korzeniowski, and H. Räcke, "A practical algorithm for constructing oblivious routing schemes," in *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, New York, NY, USA, 2003, pp. 24–33.

[10] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke, "Optimal oblivious routing in polynomial time," in *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2003, pp. 383–388.

[11] A. Bermúdez, R. Casado, F. J. Quiles, T. M. Pinkston, and J. Duato, "Evaluation of a subnet management mechanism for infiniband networks," in *32nd International Conference on Parallel Processing (ICPP 2003), 6-9 October 2003, Kaohsiung, Taiwan*. IEEE Computer Society, 2003, p. 117.

[12] M. D. Schroeder, A. Birell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker, "Autonet: A high-speed, self-configuring local area network using point-to-point links," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, Oct. 1991.

[13] O. Lysne, T. Skeie, S.-A. Reinemo, and I. Theiss, "Layered routing in irregular networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 1, pp. 51–65, 2006.

[14] T. Hoefler, T. Schneider, and A. Lumsdaine, "Optimized routing for large-scale infiniband networks," in *HOTI '09: Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 103–111.

[15] T. Skeie, O. Lysne, J. Flich, P. Lopez, A. Robles, and J. Duato, "Lash-tor: A generic transition-oriented routing algorithm," in *ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference*. Washington, DC, USA: IEEE Computer Society, 2004, p. 595.

[16] J. C. Sancho, J. Flich, A. Robles, P. L. López, and J. Duato, "Analyzing the influence of virtual lanes on the performance of infiniband networks," in *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*. Washington, DC, USA: IEEE Computer Society, 2002, p. 72.

[17] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[18] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony, "The PERCS High-Performance Interconnect," in *Proceedings of 18th Symposium on High-Performance Interconnects (Hot Interconnetcs 2010)*. IEEE, Aug. 2010.

[19] D. B. Johnson, "A note on dijkstra's shortest path algorithm," *J. ACM*, vol. 20, no. 3, pp. 385–388, 1973.

[20] E. G. Coffman, M. Elphick, and A. Shoshani, "System deadlocks," *ACM Comput. Surv.*, vol. 3, no. 2, pp. 67–78, 1971.

[21] L. Schwiebert, "Deadlock-free oblivious wormhole routing with cyclic dependencies," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 865–876, 2001.

[22] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[23] T. Schneider, T. Hoefler, and A. Lumsdaine, "Orcs: An oblivious routing congestion simulator," Indiana University, Tech. Rep. 675, Feb. 2009.

[24] S. R. Öhring, M. Ibel, S. K. Das, and M. J. Kumar, "On generalized fat trees," in *IPPS '95: Proceedings of the 9th International Symposium on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 1995, p. 37.

[25] D. Li, X. Lu, and J. Su, "Graph-theoretic analysis of kautz topology and dht schemes," in *Network and Parallel Computing, IFIP International Conference, NPC 2004, Wuhan, China, October 2004, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3222. Springer, 2004, pp. 308–315.

[26] F. Petrini and M. Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Proceedings of the 11th International Parallel Processing Symposium, IPPS'97*, Apr. 1997, pp. 87–93.

[27] T. Hoefler, T. Mehlan, A. Lumsdaine, and W. Rehm, "Netgauge: A network performance measurement framework," in *High Performance Computing and Communications, Third International Conference, HPCC 2007, Houston, USA, September 26-28, 2007, Proceedings*, vol. 4782. Springer, Sep. 2007, pp. 659–671.

[28] G. Juckeland, S. Börner, M. Kluge, S. Kölling, W. Nagel, S. Pflüger, H. Röding, S. Seidl, T. William, and R. Wloch", "Benchit – performance measurement and comparison for scientific applications," in *Parallel Computing - Software Technology, Algorithms, Architectures and Applications*, ser. Advances in Parallel Computing, F. P. G.R. Joubert, W.E. Nagel and W. Walter, Eds. North-Holland, 2004, vol. 13, pp. 501–508.

[29] D. Bailey, T. Harris, W. Saphir, R. V. D. Wijngaart, A. Woo, and M. Yarrow, "The nas parallel benchmarks 2.0," NASA Ames Research Center, Tech. Rep. NAS-95-020, Dec. 1995.