Static Routing in Symmetric Real-Time Network-on-Chips

Florian Brandner
Embedded Systems Engineering Section
Inst. for Informatics and Mathematical Modeling
Technical University of Denmark
flbr@imm.dtu.dk

Martin Schoeberl
Embedded Systems Engineering Section
Inst. for Informatics and Mathematical Modeling
Technical University of Denmark
masca@imm.dtu.dk

ABSTRACT

With the rising number of cores on a single chip the question on how to organize the communication among those cores becomes more and more relevant. A common solution is to use a network-on-chip (NoC) that provides communication bandwidth, routing, and arbitration among the cores. The use of NoCs in real-time systems is problematic, since the shared network and all cores connected to it have to be analyzed to derive time bounds of real-time tasks.

We propose to use a *statically* scheduled, time-division-multiplexed NoC design that allows a decoupled analysis of individual real-time tasks. Our network provides *virtual circuits* between all cores. These virtual circuits are implemented by delivering messages periodically on a static, fixed routing schedule. Since the routing does not change, it can be pre-computed offline.

This work focuses on the computation of routing schedules for symmetric NoC topologies, e.g., torus and hypercube. Due to the symmetry, the all-to-all communication can be modeled via simplified *communication patterns* that are concurrently processed by all routers. The scheduling problem is solved by a heuristic that tries to maximize the overlap of active patterns. Our experiments show that, for larger networks, our heuristic yields schedule lengths that are only 15% to 20% longer than theoretical lower bounds.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems;
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Circuit-switching networks*

General Terms

Theory, Algorithms, Measurement

Keywords

Static Scheduling, Network-on-Chip, Real-Time Systems

RTNS'12 , November 08 - 09 2012, Pont a Mousson, France

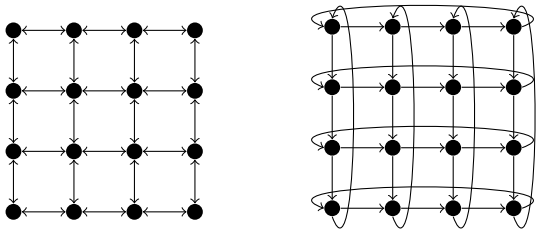
1. INTRODUCTION

In real-time systems we need analyzable processors, memories, and interconnects to perform static worst-case execution time (WCET) analysis. In a chip-multiprocessor the arbitration for shared resources (memory and communication channels) has to be static to be analyzable. In this paper we explore the generation of static communication schedules for a network-on-chip (NoC).

Our time-predictable NoC, called S4NoC [8], uses time-division multiplexing (TDM) to share the communication network among all computing cores, i.e., each core is granted exclusive access to some network resources for specific time periods. The packets in this network are single-word messages (also called flits). A message moves from one router to the next in a single cycle. It also has to move each cycle, as the routers provide no further buffering. Another assumption in the S4NoC is that the target core is able to consume all messages. There is no provision of flow control at the NoC-message level. If this is needed, it has to be implemented at a higher network level. The ability to consume the messages at the destination can be fulfilled by using a dedicated communication memory to store incoming data.

The S4NoC is intended as a generic, time-predictable platform for a wide range of real-time programs as is. The network thus periodically delivers messages between *all* pairs of cores, i.e., a periodic all-to-all communication schedule. This schedule provides equal communication bandwidth between any two cores in the NoC. It, furthermore, ensures that resource conflicts on routers and links are resolved, and deadlocks cannot appear during message routing. The use of such a fixed, all-to-all communication schedule is inherently time-predictable and analyzable, which enables the analysis of the timing behavior of the entire system.

Other statically scheduled NoCs [3, 7] rely on application-specific schedules that depend on the communication requirements of a specific application. Along with the communication schedule, also the hardware is customized. For instance, buffers are often required to resolve conflicts on shared network links [4, 9]. This is problematic in safety-critical systems, as changes in the communication requirements then incur fundamental changes to the system architecture and hardware design. The approach followed by the S4NoC is different. It provides a generic, all-to-all schedule that is applicable to a wide range of applications. This simplifies the validation and certification of safety-critical systems. Changes in the communication requirements of one part of the system do not affect other parts as their requirements are still guaranteed to be satisfied.



(a) 4×4 Mesh (b) 4×4 Torus

Figure 1: Nodes and interconnection links in two-dimensional network-on-chips.

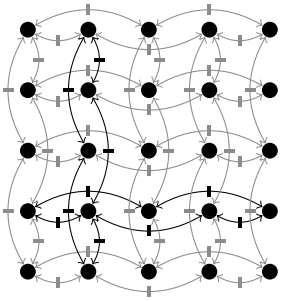


Figure 2: A 5×5 bidirectional torus using an intertwined layout and pipelining.

Besides being time-predictable and analyzable, the S4NoC design has advantages in the hardware implementation. With single-word messages it is beneficial to avoid routing information in the message itself and store the schedule tables in the routers. Without this message head, the size (and therefore bandwidth) of the NoC is freely configurable. As the schedule is static and only depends on the number of cores, the corresponding tables can literally be *hardcoded* and implemented efficiently as read-only-memory (ROM) tables in silicon. For a 4-way router, e.g., used in a bidirectional torus, each table entry encodes only 8 bits (2 bits per 4-way multiplexer). Experiments using an FPGA implementation have shown that the ROM tables are indeed very small.

We consider symmetric network topologies and search for a static schedule of individual messages to provide all-to-all communication with equal bandwidth. For such a system the intuition is that a symmetric schedule, where all routers perform the same schedule, may be an optimal schedule. The schedules found using our technique are 15% to 20% longer than theoretical lower bounds, and therefore confirm the intuition. Those schedules are found quickly, leading to a scalable approach that can be applied even to large network instances. The main contribution of this paper is a heuristic algorithm to generate these near-optimal static schedules for instances of the TDM-based S4NoC with symmetric topologies.

The implementation of the schedule construction is freely available as open source, and distributed with the hardware implementation of the NoC.[1] The generated schedules are used in an FPGA-based many-core system, where the schedule is synthesized into hardware as ROM tables. Although not in the scope of this paper, the evaluation of the derived schedules in a real prototype system gives further confidence that they are correct.

The paper is organized as follows: the next section provides background information on NoC topologies, routing, and communication schedules. Related work is presented in Section 3. Section 4 describes the scheduling algorithm in detail. The evaluation in Section 5 shows the shortest achievable scheduling periods for different topologies and NoC sizes and the paper is finally concluded in Section 6.

2. BACKGROUND

The section provides the basic background on network-on-chip topologies and how these networks can be used to provide communication from every node in the network to every other node (all-to-all).

[1]Source available at `https://github.com/t-crest/s4noc`

2.1 Network-On-Chip Topologies

A network-on-chip topology defines the structure of a concrete instance of a network. Typically a topology is defined in terms of computing cores that are connected to the network by the means of *routers*. The routers in turn are interconnected using *links*. We assume that every core is connected by two links, an incoming and an outgoing, to a single router. We thus treat a core and its router synonymously as a *node*.

Definition 1. *A concrete instance of a network topology is characterized by a directed graph $G = (V, E)$, where the vertices in V represent the nodes of the network (routers and cores) and the edges in E the interconnection links.*

Various network topologies have been proposed in the literature, ranging from tree-based organizations over two-dimensional mesh and torus structures to complex multi-dimensional hypercubes. Figure 1 depicts two instances of typical NoCs, a 4×4 2-D mesh and a 4×4 torus.

Definition 2. *A network topology is* regular *if the routers are placed and interconnected in a regular (multi-dimensional) grid. Thus all routers, except those at the boarder of the grid, are interconnected to their neighbors in a regular way.*

Definition 3. *A network topology is* symmetric *if the routers are placed and interconnected in a regular (multi-dimensional) grid. All routers, even those at the boarder, are interconnected to their neighbors in a regular way by wrapping around to the other side of the grid.*

The mesh network in Figure 1(a) is an instance of a regular network topology, whereas the torus in Figure 1(b) is a symmetric network topology. The layout of a torus network, as shown by Figure 1(b), may lead to problems in the hardware layout. The interconnection link closing the loop becomes very long and thus often leads to an reduced operation frequency of the network. An alternative approach, used in practice, is to intertwine the layout of the individual cycles for each row and each column as shown by Figure 2. Instead of connecting a node to its direct neighbor, a link is established to the next node in the row or column respectively – unless the end of the row/column has been reached. A link is now double the length as a link in a simple mesh organization, which may limit the maximal attainable clock frequency. This issue can be resolved by pipelining the communication over links, i.e., introducing additional registers on links, as shown by the bars on each link in Figure 2. The register can be interpreted as a simplified router with a single incoming and outgoing link.

In the context of this work we evaluate two-dimensional networks with *regular* and *symmetric* topologies as depicted by Figure 1 and 2, i.e., either a 2D-mesh, a torus, or a bidirectional torus. For the tori, we consider both, the regular topology and a variant using pipelined links. Note, however, that our approach is also applicable to other kinds of topologies – for instance hypercubes.

2.2 Routing

In order to deliver messages over the network, a routing mechanism has to be provided, i.e., a scheme that determines how messages are transmitted between the individual nodes within a network.

The route for a specific message can be specified by its source node, its destination node, and all intermediate links that the message traverses on the way from the source to the destination. Since we are dealing with regular and symmetric networks only, it is sufficient to encode the *direction* of the next neighbor node that the message will be forwarded to instead of concrete links.

DEFINITION 4. *A route of a message is defined by a triple $r = (s, p, d)$, where s specifies the source, d the destination node, and $p \in \mathcal{F}^+$ a non-empty string over an alphabet $\mathcal{F}$. The length of the route $|r| = |p|$ denotes the number of hops it takes to transmit the message. A route can also be interpreted as a function $r\colon \mathbb{N}_0 \to \mathcal{F} \cup \{\perp\}$ that returns the direction of the n-th hop of the route or $\perp$ if $n \geq |r|$.*

The symbols of $\mathcal{F}$ represent the direction to the neighbor to which a message will be transmitted next on each hop. In two-dimensional networks a message can be forwarded to the neighbor to the north (`n`), to the east (`e`), to the south (`s`), or to the west (`w`), i.e., $\mathcal{F} = \{\mathtt{n}, \mathtt{e}, \mathtt{s}, \mathtt{w}\}$.

EXAMPLE 1. *A simple route in a two-dimensional network from a source node s to a destination node d could be $r = (s, \mathtt{nnw}, d)$, i.e., the message travels two hops to the north and one hop to the west. The third hop of this route is $r(2) = \mathtt{w}$, while $r(0) = \mathtt{n}$ and $r(5) = \perp$.*

We assume a fixed, *static* routing scheme where each router has a *local* routing table that determines how messages are passed further on. A message thus does *not* specify its destination; the router instead forwards the message according to the fixed schedule. The message delivery to/from cores over the network interface is similarly determined by static tables, which provide a mapping between the message's destination/source nodes and the corresponding time slots of the routing tables. The time slot together with the routing scheme determines from which node a message is sent and which node ultimately receives the message.

A fixed, static schedule greatly simplifies the routers and network interfaces, since no buffers and no dynamic arbitration logic are needed. Furthermore, assuming correct schedules, deadlock avoidance and message delivery times are guaranteed by design.

2.3 All-To-All Communication

Given an instance of a network topology and assuming static routing tables the problem now is to find a *static* communication schedule that allows *every* node in the system to send a message to *every other* node in the system within a *fixed* time period. Note that a node may send different messages to different nodes; an all-to-all schedule thus should not be confused with *broadcasting* of messages.

DEFINITION 5. *An all-to-all communication schedule $S = (R, T)$ for a network $G = (V, E)$ consists of a set of routes R, such that for all pairs $s, d \in V$, $s \neq d$ there exists exactly one route $r = (s, p, d) \in R$. The function $T\colon R \to \mathbb{N}_0$ determines at which time instant a message is transmitted using a specific route.*

A feasible communication schedule, in addition, has to respect that every communication link may be used at most *once* at every time instant. Similarly, it has to ensure that every node may receive only *one* message and send only *one* message at once. *All* incoming messages at a router *have* to be forwarded to some outgoing link, i.e., messages cannot stall on the path from their source to their destination.

The communication schedule outlined here corresponds to a *circuit-switched*, *time-division-multiplexed* network that provides *virtual circuits* offering equal bandwidth between any two nodes in the system.

The problem of finding a feasible communication schedule can be characterized as a *multi-commodity flow over time* [2], where a number of commodities has to be shipped over a network in a given time period. Each commodity has to be delivered from a corresponding source to a destination such that *capacity constraints* of all links in the network are respected at all times. Based on this model various questions can be asked with respect to the flow of the commodities. For instance, one can ask for the maximal flow in this network or for the *quickest multi-commodity flow*, i.e., the shortest period in which all commodities reach their destination. A solution to the latter problem corresponds directly to an optimal schedule for the all-to-all communication problem.

3. RELATED WORK

The scheduling problem considered in this work can formally be stated as a *dynamic multi-commodity flow* problem over time. A seminal work by Ford and Fulkerson introduced *time-expanded flow networks* to model dynamic flow problems using equivalent static problems [2]. A time-expanded network is a structure containing replications of the network for several time instants (e.g., clock ticks). Fleischer and Skutella study variants of the NP-hard *quickest multi-commodity flow* problem [1] and present a polynomial 2-approximation algorithm. Although closely related, these results apply to general multi-commodity flow problems, where fractional solutions are acceptable. In the context of this work, however, integer solutions are required since the physical hardware resources are indivisible.

The S4NoC [8] is an implementation of the NoC design assumed in this work. The scheduling problem is solved optimally for small network sizes using a multi-commodity flow over time and ILP. Even though the approach does not scale – scheduling a network with 25 nodes takes several weeks – the computed optimal solutions provided important insights to this work. Most notably the benefits of symmetric network topologies, and in particular the bidirectional torus, as well as the pattern-based scheduling strategy are motivated by experiments using the optimal ILP formulation.

Several other NoCs use TDM-based routing of messages, where the communication requirements are typically specified as a set of *virtual circuits* (VC) annotated with bandwidth requirements.

Tadpole [5] is a scheduling algorithm for NuMesh. The algorithm is based on time-expanded networks and tries to

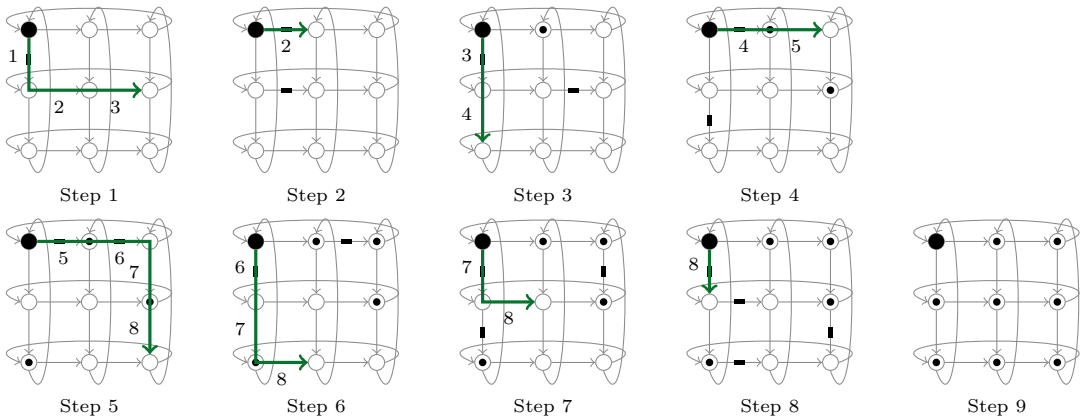


Figure 3: Routes selected by an optimal schedule (ILP), starting from the node in the top left corner. On each step a new route is highlighted. Blocked links and reached nodes are marked (bars and dots).

greedily schedule VCs according to bandwidth requirements. Congestion is modeled using a reduced graph of the network, where edge weights represent congestion. Backtracking allows the rerouting of communication due to congestion. Our approach is closely related. However, by exploiting the symmetry of the network topology and pattern-based scheduling the problem search space is drastically reduced. The algorithm thus scales even for very large problem instances, while achieving near-optimal results.

Æthereal [3] relies on a TDM-scheduling algorithm in two phases [4]: (1) path allocation and (2) TDM-slot allocation. Path allocation explores viable paths through the NoC using an adapted shortest path search capturing link congestion and bandwidth requirements. Slot allocation then tries to find a feasible mapping of a path to TDM slots satisfying bandwidth and latency constraints. The approach is driven by the path selection, which may lead to additional link buffers and an increased schedule length during slot allocation. This also applies to an improved variant of the slot allocation algorithm [9]. Our algorithm similarly proceeds in two phases, but is driven by the reservation of TDM slots during the scheduling phase. This leads to more compact and shorter schedules that do not require any additional buffers associated on the network links. This is highly important as shorter schedules increase the bandwidth between pairs of nodes and reduce the maximum transfer latency when analyzing the worst-case timing of the system.

Lu and Jantsch [6] propose a similar two-phase approach consisting of path selection and slot allocation for the Nostrum NoC [7]. They exploit the properties of *logical networks* to explore all feasible paths and slot allocations recursively in a search tree. To keep the scheduling time practical, branches of the tree are pruned eagerly be testing whether an assignment of VCs to logical networks is feasible. The two phases of the algorithm are tighter coupled as in the case of Æthereal [4], avoiding the useless insertion of additional buffers at the expense of potentially exponential computation overhead.

None of the discussed algorithms is able to deliver symmetric, all-to-all communication schedules, which ensure identical schedule tables at all routers of the S4NoC [8]. The symmetry has several advantages. For one, the overhead required to implement the scheduling tables in hardware can be reduced by sharing. More importantly, the correctness of symmetric, all-to-all schedules and their implementation in hardware is trivial to verify. This is usually required for safety-critical systems.

4. PATTERN-BASED SCHEDULING

Based on experiments using an optimal integer linear programming (ILP) formulation of our scheduling problem [8] on two-dimensional network topologies, we found that the optimal solutions to our scheduling problem for regular network topologies are by themselves almost regular. For instance, the computed optimal solutions always yield direct message routes between two nodes without detours, i.e., only *shortest* routes are selected even when the network capacity would theoretically allow for detours. Furthermore, most routes follow a simple routing scheme, where a route consists of two straight segments, a horizontal followed by an optional vertical segment, or the other way around.

Example 2. *Figure 3 depicts the routes from one node (upper left) towards all other nodes, extracted from an optimal schedule of a 3×3 torus, computed using ILP. On each step a message is sent using a new route from the source node at the top left corner to some destination node (in green). In addition, network links blocked by messages on the fly are highlighted using a black bar. Once a message has reached its destination, the respective node is marked by a dot. As can be seen, the routes simply consist of two segments. For instance, at step* 7 *a message is transmitted using a route* $(s, \texttt{se}, d)$*, where* s *denotes the top-left node and* d *the node in the center of the network. The message arrives at the destination on time step* 9*, while a communication link is blocked by a message transmitted over the route at step* 8*.*

4.1 Symmetric Schedules

These observations, combined with properties of symmetric (and regular) network topologies, give rise to a simple heuristic that (1) assumes that all (or almost all) routers take the same routing decisions on every time instant and (2) that considers only simple, at most n-segmented routes, where n is a constant depending on the network topology (e.g., 2 for the two-dimensional topologies considered here). The basic idea of our approach is to abstract all the currently active routes in a schedule by a *communication pattern*, i.e.,

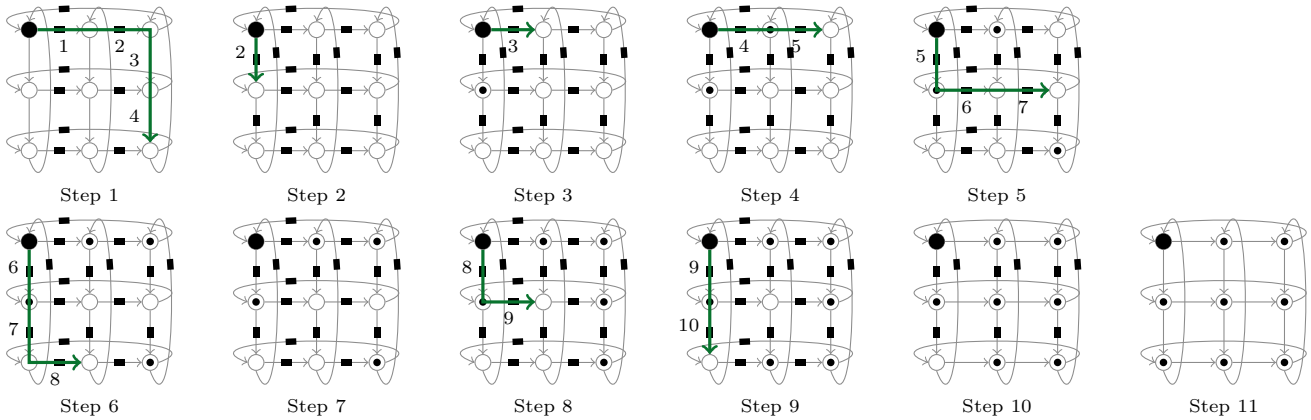


Figure 4: Symmetric schedule constructed by reordering the routes from Figure 3. Each step highlights a new route. Almost all network links are used at all times, as all routers in the network operate synchronously.

a route without a specific source and destination node. We then define a conflict relation between these communication patterns and devise an algorithm that computes an ordering of a set of communication patterns that provides a conflict-free, all-to-all communication schedule.

DEFINITION 6. *A communication schedule $S = (R, T)$ is* symmetric *when all routes scheduled at the same time instant are transmitting the respective message along the same path through the network, shifted according to the position of the route's source node, i.e., for all routes $r_1 = (s_1, p_1, d_1)$ and $r_2 = (s_2, p_2, d_2)$: $T(r_1) = T(r_2) \iff p_1 = p_2$. The path p common to all routes scheduled at a specific time instant is called a* communication pattern.

Note that the original schedule from Figure 3 is *not* a symmetric schedule, because the relative order in which messages are sent across these routes would lead to conflicts in the usage of links when the communication patterns are applied at other nodes. For instance, a message sent from a node at time step 1 would conflict with a message sent from the node's lower neighbor at time step 2. However, it is possible to use the communication patterns from the former example and reorder them to generate a symmetric schedule, as shown in the following example.

EXAMPLE 3. *Figure 4 shows a symmetric schedule for a 3×3 torus network. At each step* all *nodes in the network send a message according to the highlighted route in green (shifted accordingly). Since all nodes send messages concurrently all vertical and/or horizontal communication links are used at the same time. The schedule consists of* 8 *communication patterns* `eess`, `s`, `e`, `ee`, `see`, `sse`, `se`, *and* `ss` *that are scheduled at time instant* 1 *through* 6, 8, *and* 9 *respectively.*

Algorithm 1 Compute a symmetric and conflict-free communication schedule.

Require: $G = (V, E)$ . . . An instance of a network topology
Ensure: A symmetric, conflict-free schedule $S = (R, T)$

```
// Compute candidate communication patterns
P = Candidates(G)
// Select and schedule communication patterns
S = Schedule(P)
return S
```

DEFINITION 7. *A* symmetric *schedule $S = (R, T)$ is* conflict-free *when the following two conditions are met:*

- *For two routes $r_1, r_2 \in R, T(r_1) \neq T(r_2)$:* $T(r_1) + |r_1| \neq T(r_2) + |r_2|$,
- *For two routes $r_1, r_2 \in R, T(r_1) < T(r_2) < T(r_1)+|r_1|$,* $\forall i \in \{0, \ldots, T(r_1) + |r_1| - T(r_2)\}$*:* $r_1(T(r_2) - T(r_1) + i) \neq r_2(i)$

The definition from above ensures that (1) no two routes start or end at the same time instant in the communication schedule and (2) that any two routes scheduled concurrently utilize disjoint sets of communication links.

Since routes and communication patterns can be treated interchangeably in a symmetric schedule, we can devise an algorithm that, given a set of communication patterns, computes a conflict-free, symmetric schedule. The algorithm consists of two phases. Firstly, a set of communication patterns is computed that provides at least one pattern for any pair of nodes. Secondly, a schedule is constructed by iteratively selecting candidate patterns and assigning them a time slot in the schedule, while avoiding conflicts. Algorithm 1 provides an overall view of our heuristic.

4.2 Candidate Patterns

The first step of our heuristic is to compute candidate patterns, which can be used to construct the actual communication schedule. In contrast to traditional, static routing algorithms, which often call this phase *path selection*, a candidate pattern is not necessarily included in the final result, e.g., if another candidate providing an equivalent route has been scheduled instead. An important property of this phase is to ensure that the set of candidate patterns contains all patterns needed to provide all-to-all communication. Algorithm 2 computes such a candidate set.

Algorithm 2 `Candidates`(G) – Compute a candidate set of communication patterns.

Require: n . . . Max. number of route segments
$G = (V, E)$ . . . An instance of a network topology
Ensure: A set of candidate patterns P

```
for all v1, v2 ∈ V, v1 ≠ v2 do
    All shortest max. n-segmented routes (v1, p, v2): p ∈ P
return P
```

Algorithm 3 $\texttt{Candidates}_{\text{Tous}}(G)$ – Compute candidate communication patterns for regular torus topologies.

Require: $G = (V, E)$ ... An instance of a network topology
Ensure: A set of candidate patterns P

```
1: let o be the top-left most node in the grid defined by G
2: for all v ∈ V, v ≠ o do
3:     All shortest max. 2-segmented routes (o, p, v): p ∈ P
4: return P
```

The algorithm is kept rather generic and might compute the same communication patterns over and over again. This can be avoided by exploiting the structure of the network topology in question. In symmetric topologies it is sufficient to enumerate all n-segment routes from one specific *origin* node to all other nodes (due to the wraparound). For instance, it is sufficient to consider only the routes from the top-left most node (chosen for convenience) in a regular torus topology to compute all desired patterns (see Algorithm 3). A similar optimization can also be done for bidirectional tori - with the only exception that it is more convenient to chose a node at the center of the grid as origin e.g., at coordinates $(\lceil \frac{m}{2} \rceil, \lceil \frac{m}{2} \rceil)$ with respect to the grid defined by an m×m-bidirectional torus.

EXAMPLE 4. *Consider the 3×3 torus as shown by Figure 4. The candidate set for this network, consisting of the shortest 2-segment routes form the top-left node (marked by a filled circle in the figure) to any other node, is given by* {e, ee, s, es, se, ees, see, ss, ess, sse, eess, ssee}. *If it were a bidirectional torus the candidate set would be* {wn, nw, n, en, ne, w, e, ws, sw, s, es, se}.

During the computation of the candidate set, it is trivial to track classes of *equivalent* communication patterns, i.e., all those patterns that deliver a message from a given source node in the NoC to the *same* destination node.

DEFINITION 8. *The* equivalence class *of a communication pattern $p \in P$, denoted by the function $EC\colon P \to \mathcal{P}(P)$, consists of all communication patterns in P that, given a specific source node in the NoC, deliver a message to the same destination node.*

4.3 Schedule Construction

After computing the set of candidate patterns the actual schedule is constructed using Algorithm 4 by iteratively performing four steps: (1) selecting a pattern from the candidate set, (2) finding a suitable time slot for the candidate within the partial schedule constructed so far, (3) appending corresponding routes to the final schedule, and (4) removing all communication patterns equivalent to the selected candidate form the candidate set.

On every iteration, a candidate is selected to be scheduled next, using the `SelectCandidate` function. In order to keep the algorithm simple (and reasonably fast), the schedule constructed so far is not taken into account during the candidate selection. However, we consider a heuristic that tries to prevent conflicts with the candidate selected in the preceding iteration of the algorithm. For our experiments (Section 5) we evaluate four different selection strategies:

- Selection of a random pattern of the candidate set:
  $\texttt{SelectCandidate}_{\text{Rnd}}(P) = \text{random } p \in P$

Algorithm 4 $\texttt{Schedule}(P)$ – Derive a symmetric, conflict-free, all-to-all communication schedule from a set of candidate patterns.

Require: P ... A set of candidate patterns
$G = (V, E)$ ... An instance of a network topology
Ensure: A symmetric and conflict-free schedule $S = (R, T)$

```
1: while P not empty do
2:     // Find a good candidate pattern
3:     let p = SelectCandidate(P)
4:     // Find time slot
5:     let t = ScheduleCandidate(R, T, p)
6:     // Append routes to schedule
7:     for all v ∈ V do
8:         Construct a route r starting at v using p
9:         Append r to R
10:        Assign r to time step t via T
11:    // Remove equivalent patterns
12:    remove all patterns in EC(p) from P
13: return S = (R, T)
```

- Selection of any of the shortest candidate patterns:
  $\texttt{SelectCandidate}_{\text{Sht}}(P) = \text{random } p \in P \text{ s.t. } \nexists q \in P, |q| < |p|$
- Selection of any of the longest candidate patterns:
  $\texttt{SelectCandidate}_{\text{Lng}}(P) = \text{random } p \in P \text{ s.t. } \nexists q \in P, |q| > |p|$
- Selection of the longest candidate pattern avoiding conflicts with p', the candidate of the *previous* iteration:
  $$\texttt{SelectCandidate}_{\text{Cnfl}}(P) = \text{random } p \in P \text{ s.t.}$$
  $$\nexists q \in P\colon |q| > |p| \wedge$$
  $$(\nexists i, j \in \mathbb{N}_0\colon p(i) = p'(j) \vee$$
  $$\forall r \in P, |r| = |p|\colon \exists i, j \in \mathbb{N}_0\colon r(i) = p'(j))$$

The selected candidate is then scheduled at the *earliest* time instant that is not yet assigned to another communication pattern of a previous iteration and that avoids all conflicts with all the communication patterns scheduled so far. Conflicts are easily determined for every possible time instant according to the conflict relation as given by Definition 7. More efficient conflict-detection schemes that track available time slots are possible, but are not considered in this work. More formally the scheduling function is defined as follows:

$$\texttt{ScheduleCandidate}(R, T, p) = \text{minimal } t \in \mathbb{N}_0 \text{ s.t.}$$
$$\nexists r \in R\colon T(r) = t \wedge$$
$$\nexists r \in R, T(r) \neq t\colon t + |p| = T(r) + |r| \wedge$$
$$\nexists r \in R, t < T(r) < t + |p|, i \in \{0, \ldots, t + |p| - T(r)\}\colon$$
$$p(T(r) - t + i) = r(i) \wedge$$
$$\nexists r \in R, T(r) < t < T(r) + |r|, i \in \{0, \ldots, T(r) + |r| - t\}\colon$$
$$r(t - T(r) + i) = p(i)$$

Routes are then constructed for every node in the network using the selected candidate pattern. These routes are appended to the final schedule and assigned to the previously determined time slot. Note that in *regular, non-symmetric* networks the boarder of the grid defined by the network has

to be accounted for, i.e., routes are not constructed for a specific source node in case the communication pattern would reach out over the boarder of the grid.

Finally, all patterns that are in the equivalence class of the selected candidate pattern are removed from the candidate set in order to avoid duplicated routes. The result is a symmetric, conflict-free, all-to-all communication schedule.

EXAMPLE 5. *Assuming the longest candidate selection strategy and the candidate set of the 3×3 torus in Example 4, Algorithm 4 proceeds as follows: initially the entire candidate set is available in P and the communication schedule is empty. The selection strategy randomly chooses to schedule one of the longest patterns in P, say* `eess` *(l. 3). Since the schedule is empty, the pattern can be scheduled at time instant* 0 *(l. 5) and corresponding routes are added to the communication schedule for every node in the network (l. 8). Next,* `eess` *and* `ssee` *are removed from P since both are in the equivalence class $EC($*`eess`*$)$ (l. 11), i.e., given a source the same destination is reached by both patterns. In the next iteration, the algorithm selects the* `see` *pattern, which can be scheduled at time instant* 4 *the earliest due to conflicts (Definition 7). Along with this pattern also the equivalent* `ees` *pattern will be removed from P. The algorithm then continues to select patterns, append routes to the communication schedule, and eliminate equivalent patterns by scheduling* `sse`*,* `se`*,* `ee`*,* `s`*, and* `e` *at time instant* 5*,* 1*,* 3*,* 8*, and* 9*.*

4.4 Correctness

This section covers correctness considerations of our pattern-based approach in general as well as considerations with regard to the heuristic scheduling algorithm.

LEMMA 1. *A symmetric, conflict-free, all-to-all communication schedule for a symmetric or regular network is a feasible all-to-all communication schedule (see Section 2.3).*

PROOF. A symmetric, all-to-all schedule is an all-to-all communication schedule. Furthermore, the definition of routes (see Definition 4) does not allow us to specify stalling messages, i.e., all incoming messages at the router of a node are immediately forwarded on the next time instant.
Recalling Definition 6 and 7, we thus only have to show that at most one message is transmitted over a link in the network at any moment in time, i.e., feasibility. For this we have to consider three cases:
Sending: The link between a sending node and its router is used at most once, since, following Definition 6, a single communication pattern is permitted to commence on every time instant only.
Receiving: The link between a receiving node and its router is used only once, due to Definition 7.
Routing: A link between the routers of two nodes is used only once, due to Definition 7. □

THEOREM 1. *Algorithm 1 yields a symmetric, conflict-free, all-to-all schedule for symmetric or regular networks.*

PROOF. We will informally sketch a proof for each of the three properties:
Symmetry: The schedule is symmetric since all routes assigned to a time instant are constructed using the same candidate pattern (Algorithm 4, l. 6), no other pattern can be assigned to the same time instant (see `ScheduleCandidate`), and a pattern can only be scheduled once (l. 11).

		#links				
		mesh	torus		bid. torus	
side-length	#nodes	–	r.	p.	r.	p.
3	9	24	18	36	36	72
4	16	48	32	64	64	128
5	25	80	50	100	100	200
6	36	120	72	144	144	288
7	49	168	98	196	196	392
8	64	224	128	256	256	512
9	81	288	162	324	324	648
10	100	360	200	400	400	800
11	121	440	242	484	484	968
12	144	528	288	576	576	1152
13	169	624	338	676	676	1352
14	196	728	392	784	784	1568
15	225	840	450	900	900	1800

Table 1: Overview of network topology variants and sizes (r. and p. denote regular and pipelined variants respectively).

Conflict freedom: This follows immediately from the definition of `ScheduleCandidate`.
All-to-all communication: Assume two nodes s and d of the network for which there exists *no* route from s to d in the final schedule. It is easy to see that the function `Candidates` computes at least one communication pattern for every pair of nodes in the network – in particular, the candidate set also contains a pattern p suitable for a route $r = (s, p, d)$. Since r has not been scheduled some other *equivalent* pattern p' must have been scheduled, causing the removal of p from the candidate set (see Algorithm 4, l. 11). However, this implies the existence of a route (s, p', d) in the schedule, which contradicts our initial assumption. □

As indicated in Section 4.2, it is not necessary to compute the patterns for all pairs in symmetric networks, due to the wraparound at the boarder of the grid defined by the network. The preceding proof can easily be adopted in order to account for this optimization by showing that the resulting patterns still ensure all-to-all communication.

5. EXPERIMENTS

We evaluated our heuristic algorithm to compute communication schedules for various instances of the mesh, torus, and bidirectional torus topologies over a square grid (see Section 2.1) with a side length between 3 to 15 nodes. In our experiments we thus consider networks consisting of between 9 and 225 nodes interconnected by between 18 and 1800 links – see Table 1 for an overview. According to our experiments, the bidirectional torus topology appears to be the most interesting network topology, we thus considered even larger network configurations of up to 900 nodes and 3600 links. For the torus and bidirectional torus topology we considered both, the regular and the pipelined variants.

All measurements were performed on an AMD Fusion E-350 (stepping 0) dual-core processor running at 1.6 GHz; having a 2×512 KiB L2-cache and 2 GiB of main memory. We used a 64-bit, Linux-based operating system with kernel version 2.6.37.6 (OpenSuse 11.4). Time measurements were taken over several runs on an otherwise unloaded machine.

Note that the experimental setup was designed to evaluate the scheduling algorithm. The following numbers thus do not cover hardware requirements, e.g., to synthesize the

	#patterns		
#nodes	mesh	torus	bid. torus
9	40	12	12
16	84	24	24
25	144	40	40
36	220	60	60
49	312	84	84
64	420	112	112
81	544	144	144
100	684	180	180
121	840	220	220
144	1012	264	264
169	1200	312	312
196	1404	364	364
225	1624	420	420

Table 2: Number of candidate patterns enumerated for various network topologies.

schedule tables, nor measurements of the network utilization by specific applications, e.g., load, latency, or throughput.

5.1 Candidate Enumeration

The first phase of our approach (Algorithm 2 from Section 4.2) computes candidate patterns that are considered for the construction of the final communication schedule. The number of those patterns depends on the network topology and its *symmetry*. In our experiments we only consider 2-segment routes, which consist of a horizontal segment followed by an optional vertical segment, or the other way around. Note that also other kinds of candidate patterns could be considered. For the symmetric torus topology the number of candidate patterns is moderately increasing with the number of nodes in the network from 12 to 420 – as shown by Table 2. Note that pipelining on the links is only relevant to the length of the enumerated patterns but not their number, i.e., regular as well as the pipelined variants result in the same number of candidate patterns. Even though the bidirectional torus offers twice the number of links compared to the torus topology the number of candidate patterns remains the same for networks with the same number of nodes. This is explained by the fact that the patterns induced by the additional links are redundant. In addition, due to the bidirectional connectivity, the maximal length of the shortest routes between any two nodes is reduced, which leads to even more redundant patterns. The opposite is true for the mesh topology. Here many more candidate patterns have to be enumerated, because of the missing links compared to a torus, which would wrap around the grid. The number of candidates thus is between 40 and 1624.

The average length over all enumerated candidates is, in general, close to the network's side length for all topologies. The pipelined variant of the torus topology and the regular bidirectional torus are notable exceptions. In the former case, the average length of candidate patterns is twice the side length. More interestingly, the average length for the bidirectional torus is only half the side length, which leads to much shorter schedules.

5.2 Schedule Construction

The next phase, corresponding to Algorithm 4 of Section 4.3, derives a symmetric, all-to-all communication schedule by iteratively selecting one of the candidate patterns and assigning it to a time slot. The length of the resulting schedule depends, among others, on the order in which candidate patterns are chosen. We thus compared several heuristics for the candidate selection function: Rnd, Sht, Lng and Cnfl, which select a random candidate, one of the shortest patterns, one of the longest patterns, or one of the longest patterns without conflict respectively.

	#cycles								
	mesh			torus			bid. torus		
#nodes	heur.	opt.	rat.	heur.	opt.	rat.	heur.	opt.	rat.
9	28	10	2.80	12	11	1.09	11	10	1.10
16	59	18	3.30	28	26	1.08	20	18	1.11
25	112	34	3.29	57	52	1.10	28	28	1.00
64	481	–	–	246	–	–	88	–	–
100	974	–	–	501	–	–	158	–	–
225	3467	–	–	1821	–	–	481	–	–
400	–	–	–	–	–	–	1164	–	–

Table 3: Best schedule lengths for some selected configurations (heur.), optimal solutions (opt.), and ratio (rat.) between them (in cycles, lower is better).

Table 3 summarizes the schedule length in cycles for a selection of network configurations and compares the result to results derived using an optimal ILP formulation. The optimal schedules are available for all considered network topologies up to a size of 25 nodes. The schedule lengths for the regular mesh topology are within a factor of 3.30 with regard to the optimal solution. This is hardly surprising, since some of the available link capacity is wasted by the intrinsic assumption of our heuristic that all routers perform the same actions at all times. The situation changes drastically for torus-based networks. Our heuristic reaches the optimum for one case and generally is within 10% of the optimum. The results for the bidirectional torus are particularly interesting, since the schedule lengths remain relatively low, even for network configurations with hundreds of nodes. For instance, the schedule length in a bidirectional torus with 100 nodes is only 158 cycles long, i.e., the schedule length is only about 1.6 times the number of nodes.

Due to the high complexity of the scheduling problem, which is NP-complete in the general case, optimal numbers are only available up to a network size of 25 nodes. We thus additionally consider analytical lower bounds as a reference. The length of communication schedules can be bounded using three metrics: (1) a bound induced by the I/O bandwidth at each node (2) a bound based on the bisection bandwidth and (3) a bound based on the capacity of the network. The I/O bound for the topologies considered here amounts to $n-1$, where n denotes the number of nodes in the network. The bisection bandwidth is an important metric in network design that specifies the minimal bandwidth that is available for the transmission of messages between any partitioning of the network nodes in two equal-sized sets. Since we are seeking an all-to-all communication schedule, the number of messages transmitted between the two node sets equals $(n/2)^2$. The bisection bound is then given by dividing this number by the number of links connecting the two sets. For the torus topology this yields $m^3/4$, where m denotes the side length of the grid defined by the network's layout (for even m). Finally, the capacity bound is given by dividing the total sum of all minimal-length routes between

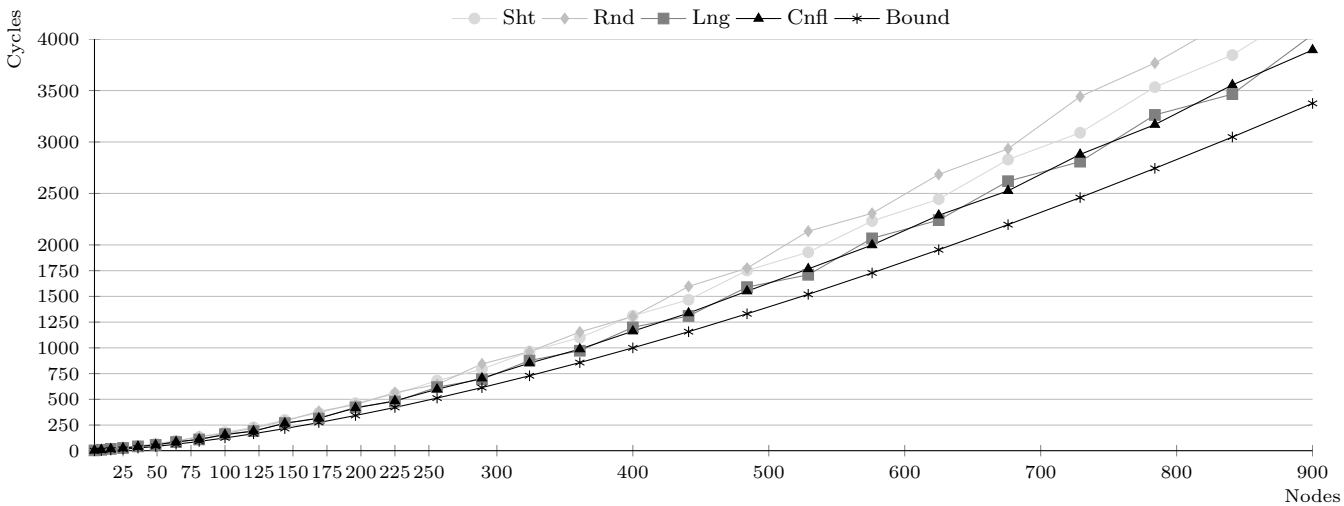


Figure 5: Schedule length and lower bound for the regular bidirectional tori (in cycles, lower is better).

any two nodes through the total number of links in the network, i.e., the network's capacity. The torus topology, for instance, has a capacity bound of $(m^3 - m^2)/2$. We denote the maximum of the listed bounds as *the* lower bound for the remainder of this discussion. As a general rule, this lower bound is dominated by the bisection bound for the mesh topology, while it is dominated by the capacity bound for the torus topology. In the case of the bidirectional torus the capacity and bisection bounds are virtually identical. Here, however, for network sizes of up to 49 nodes the I/O bound dominates. The bounds for torus and bidirectional torus apply to both, the regular and pipelined variant.

For larger instances of the symmetric topologies, our heuristic approach performs surprisingly well. For the regular torus the cnfl candidate selection clearly performs best, yielding schedule lengths within about 15% to the lower bound (see Table 4). The lng approach gives slightly inferior results. The two other methods (rnd and sht) perform worse, yet reach the lower bound within 23% to 24%. In the case of the bidirectional torus topology we observe a similar trend, the cnfl and lng candidate selection clearly perform best. This time, however, we see an alternating behavior. For network instances with an even number of nodes the cnfl approach performs better, for odd numbers lng performs best. This pattern continues consistently up to the maximal network sizes considered in our experiments (900 nodes). As of now, we do not have a conclusive explanation for this behavior.

Figure 5 and 6 illustrate the development of the schedule length, as computed by our heuristics, with increasing network size for the bidirectional torus and regular torus. The bidirectional torus allows for considerably shorter schedules than comparable torus networks. The curve follows a very gentle, though non-linear slope. For instance, in a network consisting of 225 nodes, our heuristic yields a schedule length of 486 versus 1820 cycles for the bidirectional torus and torus respectively (both within 15% to their respective lower bounds). The torus thus gives a schedule that is about 3.75 times longer than that of a bidirectional torus. This gap is

	torus				bid. torus			
#nodes	rnd	sht	lng	**cnfl**	rnd	sht	lng	**cnfl**
9	1.11	1.33	1.11	**1.11**	1.12	1.25	1.25	**1.12**
16	1.21	1.21	1.08	**1.08**	1.20	1.27	1.27	**1.20**
25	1.26	1.18	1.10	**1.10**	1.25	1.38	1.12	**1.08**
36	1.16	1.18	1.10	**1.09**	1.23	1.37	1.23	**1.17**
49	1.20	1.18	1.11	**1.08**	1.35	1.44	1.19	**1.19**
64	1.15	1.19	1.12	**1.09**	1.42	1.55	**1.33**	1.34
81	1.22	1.20	1.13	**1.10**	1.53	1.48	**1.21**	1.23
100	1.24	1.21	1.14	**1.11**	1.34	1.45	1.30	**1.25**
121	1.22	1.22	1.16	**1.12**	1.34	1.37	**1.15**	1.16
144	1.22	1.22	1.17	**1.13**	1.34	1.39	1.25	**1.24**
169	1.23	1.23	1.17	**1.14**	1.40	1.34	**1.15**	1.16
196	1.24	1.24	1.18	**1.15**	1.31	1.35	1.23	**1.21**
225	1.23	1.24	1.19	**1.15**	1.34	1.31	**1.14**	1.15
900	–	–	–	–	1.37	1.28	1.20	**1.15**

Table 4: Schedule length in relation with its theoretical lower bound for the torus and bidirectional torus topologies (lower is better).

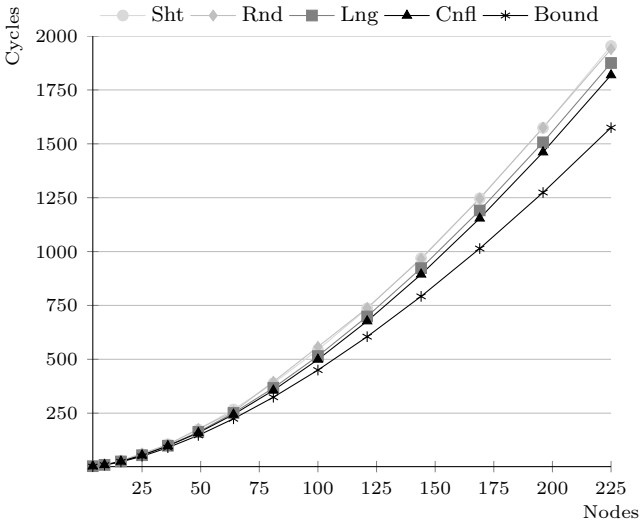


Figure 6: Schedule length and lower bound for the regular torus topology (in cycles, lower is better).

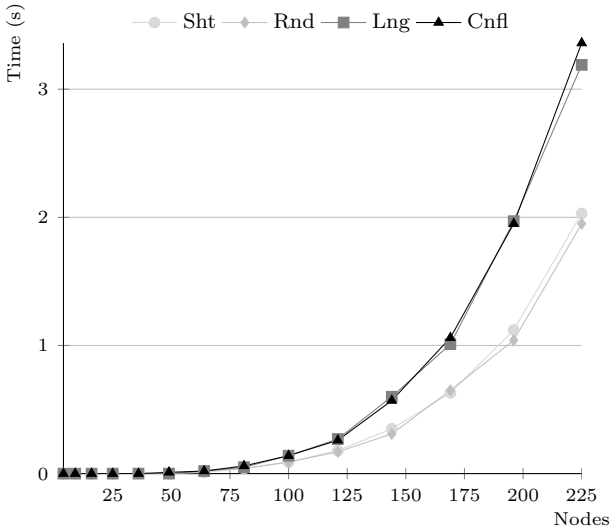


Figure 7: Execution time of the entire schedule construction for the regular bidirectional torus topology (in seconds, lower is better).

even further increasing for larger networks. The increasing gap is explained by the additional network capacity, combined with shorter candidate routes. This difference is a strong argument for a bidirectional torus: for double of the wire area the bandwidth is almost quadrupled.

Due to space considerations, we only briefly discuss pipelined network variants for the torus-based topologies. Due to the additional register on each link and the doubled length of all routes in the network, one might simply expect an increase in the schedule length by a factor of two in comparison to a corresponding network without those registers. Indeed, this intuition is not entirely wrong. As the node count increases we observe that the ration in the schedule length approaches a factor of two. For smaller instances of up to 25 nodes, the ratio ranges between a 1.33 and 1.61 for the bidirectional torus topology. This indicates that the additional registers do not help to realize any bandwidth gains, since they are counterbalanced by the increased schedule length. The additional hardware overhead, if not justified otherwise, is thus better invested elsewhere.

5.3 Execution Time

The various phases of our heuristic depend to a large degree on the underlying network topology. However, it is easy to see that the algorithm is polynomial in the number of nodes. This is also confirmed by the measured execution times for our experiments. The measurements show that the computation of candidate patterns is negligible in practice and that the total execution time is dominated by the actual schedule construction. Figure 7 shows the measured execution time for the bidirectional torus topology up to network sizes of 225 nodes. The curves for other configurations follow a similar trend with increased execution times.

6. CONCLUSION

To provide more processing power in real-time systems several processing cores are integrated on a single chip. A shared on-chip network allows communication between these cores. To avoid interference between the cores the communication on the network has to be statically scheduled.

We presented a heuristic to generate static schedules to provide all-to-all communication on symmetric network topologies, such as torus and hypercube. For symmetric topologies it is sufficient to consider only simple communication patterns that can be replicated at all routers. Therefore, the size of the search space is heavily reduced and solutions, within 15% to 20% of theoretical lower bounds, can be found even for large network instances.

The generated schedules have, furthermore, been used for an FPGA-based prototype implementation of a NoC with small micro controllers as processing cores. The schedule tables generated by our approach are directly synthesized into ROM tables and thus require minimal space.

Acknowledgments

This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

7. REFERENCES

[1] L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In *Proceedings of the Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 36–53. Springer, 2002.

[2] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[3] K. Goossens and A. Hansson. The AEthereal network on chip after ten years: Goals, evolution, lessons, and future. In *Proceedings of the Design Automation Conference (DAC)*, pages 306–311, 2010.

[4] A. Hansson, K. Goossens, and A. Radulescu. A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic. *VLSI Design*, 2007:16, 2007.

[5] P. J. LoPresti. Tadpole – an off-line router for the NuMesh system. Master's thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 1997.

[6] Z. Lu and A. Jantsch. TDM virtual-circuit configuration for network-on-chip. *IEEE Trans. Very Large Scale Integr. Syst.*, 16:1021–1034, August 2008.

[7] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone-a communication protocol stack for networks on chip. In *Proceedings of the Conference on VLSI Design*, pages 693 – 696, 2004.

[8] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, pages 152–160. IEEE, 2012.

[9] R. Stefan and K. Goossens. An improved algorithm for slot selection in the æthereal network-on-chip. In *Proceedings of the Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC)*, pages 7–10. ACM, 2011.