Static Scheduling of a Time-Triggered Network-on-Chip based on SMT Solving

Jia Huang, Jan Olaf Blech, Andreas Raabe, Christian Buckl
fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
{huang,blech,raabe,buckl}@fortiss.org

Alois Knoll
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
knoll@in.tum.de

***Abstract*—Time-Triggered Network-on-Chip (TTNoC) is a networking concept aiming at providing both predictable and high-throughput communication for modern multiprocessor systems. The message scheduling is one of the major design challenges in TTNoC-based systems. The designers not only need to allocate time slots but also have to assign communication routes for all messages. This paper tackles the TTNoC scheduling problem and presents an approach based on Satisfiability Modulo Theories (SMT) solving. We first formulate the complete problem as an SMT instance, which can always compute a feasible solution if exists. Thereafter, we propose an incremental approach that integrates SMT solving into classical heuristic algorithms. The experimental results show that the heuristic scales significantly better with only minor loss of performance.**

I. Introduction

Reliable and predictable communication is highly desirable for many embedded systems. In particular, for safety-related real-time applications, meeting real-time constraints, such as the end-to-end latency can be as important as guaranteeing functional correctness. Time-Triggered (TT) communication is a natural and efficient way to meet these requirements. In TT networks, the communication entities are synchronized with each other. Traffic is injected strictly adhering to the predefined schedule and resource collision is avoided by design. Examples of time-triggered protocols include Flexray (the static segment) in the automotive industry, SAFEBus and TTP in the avionics domain, and TTEthernet being an extension of the classical Ethernet.

The traditional time-triggered protocols usually operate on bus-like systems. The shared communication media is organized in time slots and all messages are separated in the time domain. However, bus-based systems cannot meet the communication requirements of modern Multiprocessor System-on-Chip (MPSoC) platforms [1] due to the bandwidth limitation. Researchers therefore investigated the integration of time-triggered communication in Networks-on-Chip (NoC) and proposed the Time-Triggered Network-on-Chip (TTNoC) architecture [2]. TTNoC is based on a network of on-chip switches. Although the network is globally arbitrated in time, a major advantage of TTNoC is the possibility to separate messages in the spatial domain, i.e. messages can share the same time-slot as long as their routes are non-overlapping. An example is depicted in Figure 1. In this case, TTNoC message scheduling becomes a two-fold problem: allocation of slots in the time domain and assignment of routes in the space domain.

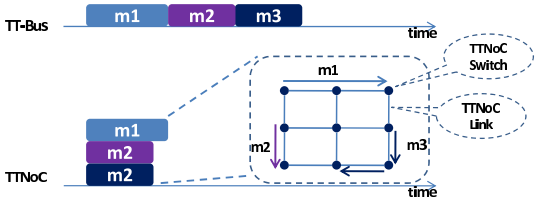

Fig. 1. Example of TTNoC Schedule

This paper tackles the TTNoC scheduling problem and presents an approach based on Satisfiability Modulo Theories (SMT) solving. An SMT solver accepts problems formulated in first-order logic and checks the feasibility of a solution with respect to background theories. We first present a specification formulating the complete problem as an SMT instance (Section III). This approach always computes a feasible solution if it exists. Since the solving time may become unacceptable as the problem size grows, we develop an incremental algorithm to improve the scalability (Section IV).

A. Related Work

The scheduling problem specific to TTNoC has not been studied in existing literature. However, our work is closely related to the scheduling approaches designed for other time-triggered architectures. In [3] the authors present an approach for the scheduling of the static segment of Flexray using Integer Linear Programming (ILP). *Lukasiewycz et al* propose a transformation of the Flexray scheduling into a bin-packing problem and solve it subsequently using ILP [4]. To increase the effective bandwidth, the concept of switched Flexray is proposed in [5]. The corresponding scheduling problem is studied in [6], [7]. Their solutions are based on the branch-and-price algorithm [6] and graph-based heuristics [7]. The scheduling problem for time-triggered multi-hop networks is considered in [8]. The author adopts an approach similar to ours. A pure SMT formulation is presented, followed by an incremental method to improve the scalability. One major difference between [8] and our work is that the route of each message is assumed to be known in [8] and therefore the author focuses only on the time domain.

II. Problem Definition and Transformation

The TTNoC architecture consists of a set of fragment switches (also called *nodes*). A switch offers four identical

978-3-9810801-8-6/DATE12/©2012 EDAA

ports as depicted in Figure 2. Each port-to-port connection consists of one link per direction (full-duplex). A port can connect to another switch or a Processing Element (PE) via the Trusted Interface Sub-System (TISS). The unified interface of switches and TISSs allows the designer to implement different topologies with low effort. A set of routes may co-exist as long as no two routes use the same link, e.g. in Figure 2, the messages m_0, m_1 and m_2 can co-exist whereas m_3 collides with m_1. The switches are not aware of the communication schedule and just forward the message from the input port to the output port according to the routing information contained in the message header. The latency of forwarding is constant.

The payload of a message is decomposed into a set of fixed-size *flits*, which is the basic transmission unit in TTNoC. A flit is handled by a switch in one *system clock cycle*. The TTNoC is globally arbitrated using TDMA. The granularity of the TDMA slots is called a *macro tick*. A macro tick is a multiple of the system clock cycle, i.e. multiple flits can be sent in one slot. The duration of a macro tick is restricted to be a negative power of a physical second by design [2], e.g. $\frac{1}{2}$ or $\frac{1}{4}$ second. The time slots are allocated statically to each communication entity and the information is stored in each TISS. The TISS abstracts the details of communication away from the application side. The TISSs are synchronized in macro tick, i.e. all communication activities are aligned to the TDMA slots. In the remainder of the paper, macro tick is used as the basic unit of time.

We focus on periodic messages in this paper. This is the typical case in the targeted application domain. A message is described as a four-tuple (s, t, p, l), where s is the message source, t is the message sink, p is the period and l is length of the message. According to the timing specification of TTNoC, the period must be a positive power of two of macro ticks ($p \in \{2^n | n > 0\}$), i.e. the messages are **harmonic**. The length l is the number of TDMA slots needed to transmit the message. To guarantee collision freedom, all flits should reach their destination before the end of the allocated time interval. Let $payload$ be the message payload in terms of flits, x be the number of hops in the route, d be the delay per hop and T be the number of flits per macro tick, the total number of TDMA slots needed by a message can be computed as $l = \left\lceil \frac{payload+xd}{T} \right\rceil$. As can be seen, the message length depends on the routing. This dependency causes a correlation between the time and space domain in the scheduling problem and it increases the complexity significantly. To cope with this problem, we introduce a restriction on routing. Let the distance (in hops) between source and the target TISS of a message m be D_m. We restrict the routing algorithm to explore routes with a maximum of $D_m + \alpha$ hops, where $\alpha \geq 0$ specifies the *flexibility* of the routing. By doing so, l can be over-estimated by $l = \left\lceil \frac{payload+(D+\alpha)d}{T} \right\rceil$.

The TTNoC scheduling problem can be stated as follows. Given an architecture with a set of nodes N and links B, a set of communicating PEs C and a set of messages M, determine: 1) the PE-to-switch allocation π, i.e. the PE c joins the network via a port of the switch $\pi(c)$, 2) the timing offset (or *phase*) f of each message, 3) the path P for each message, such that each two messages are separated either in the time or in the space domain. A message with period p and phase f occupies the time intervals $[np + f, np + f + l]$ with $n \in \mathbb{N}_0$. The path P must be a continuous route from the source TISS s to target TISS t. Since the message periods are always positive powers of two, the hyper-period of any set of messages is the longest period of all messages. We denote the hyper-period using p_{max}. Without loss of generality, it is sufficient to schedule only the first hyper-period.

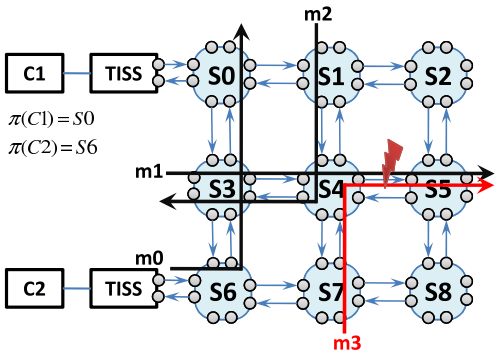

Fig. 2. The TTNoC Architecture

A. Problem Transformation

The problem of allocating messages into time intervals can be transformed into a 2D bin-packing problem. Here we adopt the transformation proposed in [4] and adapt it to our needs. A brief outline is given in the following.

Assume the shortest period of all messages in M is p_{min}. The periods of all messages can be represented as positive powers of two times p_{min}, i.e. for message m, $p_m = r_m p_{min}$, where $r_m \in \{2^n | 0 \leq n \leq p_{max}/p_{min}\}$ is the *repetition* factor. The time line of a hyper-period can be divided into *segments* of size p_{min} and viewed in a 2D fashion as shown in Figure 3a. Each message will appear in every rth segments, e.g. m_1 appears in every 1st segment and m_3 appears in every 4th segments. To transform message scheduling to bin-packing, each message is converted into a rectangle element. The size of the element can be computed by:

- $h_m = l_m$: the height of element is the length of message.
- $w_m = p_{max}/p_m$: the width of element is the number of appearances in a hyper-period.

Obviously, the widths of elements are always powers of two. The size of the bin is:

- $H = p_{min}$: the height of the bin is p_{min} in macro ticks.
- $W = p_{max}/p_{min}$: the width of the bin is the number of segments of size p_{min} in one hyper-period.

Figure 3b depicts an example. Bin-packing is about placing the elements in appropriate locations inside the bin, such that no two elements intersect. The placement of an element is defined by offsets $x_m \in [0, W)$ and $y_m \in [0, H)$ in horizontal and vertical directions. Note that the horizontal offset x_m must be a multiple of the width, i.e. $x_m \in \{nw_m | n \in \mathbb{N}_0\}$. The

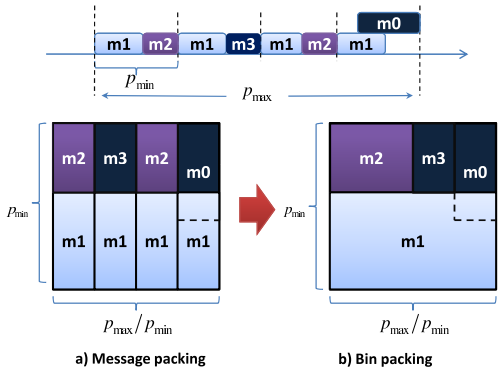

Fig. 3. Message Scheduling to Bin Packing Transformation

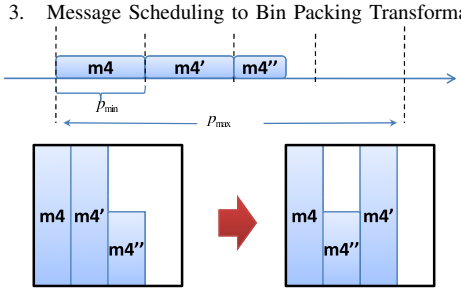

Fig. 4. Segmentation of Long Messages

placement of elements in the bin-packing problem has a one-to-one mapping to the allocation of messages in time intervals. The phase of a message m can be calculated from the position of the rectangle element as:

$$f_m = b_m p_{min} + y_m \tag{1}$$

$$b_m = t(\frac{x_m}{w_m}, \frac{W}{w_m}) \tag{2}$$

where t is the transformation function defined as:

$$t(x,y) = \begin{cases} 0 & \text{x=0} \\ t(\frac{x}{2}, \frac{y}{2}) & \text{x is even} \\ t(\frac{x-1}{2}, \frac{y}{2}) + \frac{y}{2} & \text{x is odd} \end{cases}$$

$$with \quad x \in \mathbb{N}_0, y \in \{2^n | n \in \mathbb{N}_0\}, 0 \leq x < y$$

Here b_m denotes the segment in which message m appears. The vertical position y_m denotes the offset within the segment, e.g., the offset of m_3 can be computed by $f_{m_3} = t(\frac{2}{2}, \frac{4}{1})p_{min} + h_{m_1} = p_{min} + l_{m_1}$. Using the transformation above, a feasible message schedule exists if and only if a feasible bin-packing scheme exists [4]. Since the bins are of the height p_{min}, only messages shorter than p_{min} can fit inside. Thus, if a message is longer than p_{min}, it has to be broken into several pieces. Figure 4 illustrates an example, in which m_4 occupies three time segments of size p_{min}. Those pieces can be scheduled individually. Additional constraints are needed to make sure all pieces follow the same route and are continuous in time (see Section III).

The bin-packing problem transformed from TTNoC scheduling is not a standard one. The major difference is that the intersection of objects is allowed, as long as the collision can be resolved in the space domain (e.g. m_0 and m_1 in Figure 3 can be assigned to non-overlapping routes and share the same time slot). New approach is needed the address this issue.

III. SMT Specification

This section describes the formulation of the TTNoC scheduling problem as an SMT specification. We first introduce the used variables and then proceed with the constraints that apply on the variables.

Variables. To describe the PE-to-switch allocation, we enumerate all available ports that a PE can attach to and place a *virtual component* on each port. The PEs are then mapped to the virtual components. We use $node(v)$ to denote the switch that offers the port for virtual component v. A set of binary variables is defined:

- $\mathbf{a}_{c,v} \in \{0,1\}$ is 1 iff the PE c is mapped to virtual component v.

For each message m, two sets of binary variables are used to denote the route:

- $\mathbf{q}_{m,n} \in \{0,1\}$ is 1 iff switching node n is on the path of message m and 0 otherwise.
- $\mathbf{k}_{m,i,j} \in \{0,1\}$ is 1 iff link (i,j) is on the path of message m and 0 otherwise.

The following variables specify the location of the message in the bin:

- $\mathbf{x}_m \in \{nw_m | n \in \mathbb{N}_0, n < \frac{W}{w_m}\}$ is the horizontal offset of m.
- $\mathbf{y}_m \in \{y \in \mathbb{N}_0, y < H\}$ is the vertical offset of m.

Path Constraints. We introduce a set of constraints to make sure that the variables $\mathbf{q}$ and $\mathbf{k}$ denote a continuous, acyclic path from the source to the destination. If a node is on such a path, exactly one of its input links and exactly one of its output links should be used (see Figure 2 for example). Hence, the general path constraints are:

$$\forall n \in N : \mathbf{q}_{m,n} \rightarrow (one\_in \wedge one\_out) \quad \text{where}$$

$$one\_in = (\sum_{i \in in(n)} \mathbf{k}_{m,i,n}) = 1$$

$$one\_out = (\sum_{j \in out(n)} \mathbf{k}_{m,n,j}) = 1$$

Here $out(n)$ is the set of switches reachable from the output links of n, and $in(n)$ is the set of switches that the input links of n originate from. To guarantee that the path starts and ends at the correct nodes, the following constraints need to be enforced:

- The switch that attaches the message source s_m and the one that attaches the message target t_m must be in the path of the message.

$$\mathbf{a}_{s_m,v} = 1 \rightarrow \mathbf{q}_{m,node(v)} = 1$$

$$\mathbf{a}_{t_m,v} = 1 \rightarrow \mathbf{q}_{m,node(v)} = 1$$

- Also, the link that connects the source/target of a message to the network has to be used:

$$\mathbf{a}_{s_m,v} = 1 \rightarrow \mathbf{k}_{m,v,node(v)} = 1$$

$$\mathbf{a}_{t_m,v} = 1 \rightarrow \mathbf{k}_{m,node(v),v} = 1$$

If a link is on the path, the nodes on the two ends must also be on the path:

$$\forall(i,j) \in B : \mathbf{k}_{m,i,j} = 1 \rightarrow \mathbf{q}_{m,i} = 1 \wedge \mathbf{q}_{m,j} = 1$$

Dummy loops that go from node i to j and immediately back should be avoided:

$$\forall(i,j) \in B : \mathbf{k}_{m,i,j} = 1 \rightarrow \mathbf{k}_{m,j,i} = 0$$

The overall route length should be below the upper bound:

$$\forall v_1, v_2 : (\mathbf{a}_{s_m,v_1} = 1) \wedge (\mathbf{a}_{t_m,v_2} = 1) \rightarrow \sum_{(i,j)\in B} \mathbf{k}_{m,i,j}$$
$$\leq distance(node(v_1), node(v_2)) + flexibility$$

Non-Overlapping constraints. If two messages intersect in the bin-packing, non-overlapping routes must be assigned to them. This constraint is denoted as following:

$$\forall m_1 \in M, m_2 \in M, m_1 \neq m_2 :$$
$$overlap(m_1, m_2) \rightarrow \bigwedge_{(i,j)\in B} \neg(\mathbf{k}_{m_1,i,j} = 1 \wedge \mathbf{k}_{m_2,i,j} = 1)$$

where the overlap occurs if and only if:

$$overlap(m_1,m_2) =$$
$$(\mathbf{x}_{m_1} < \mathbf{x}_{m_2} + w_{m_2}) \wedge (\mathbf{x}_{m_2} < \mathbf{x}_{m_1} + w_{m_1})$$
$$\wedge\ (\mathbf{y}_{m_1} < \mathbf{y}_{m_2} + h_{m_2}) \wedge (\mathbf{y}_{m_2} < \mathbf{y}_{m_1} + h_{m_1})$$

Problem Specific Constraints. The SMT formulation can also incorporate problem specific constraints. For example, in the current specification of TTNoC, the TISS can only transmit or receive one message at a time. This means two messages with the same source or target must be separated in time. This constraint can be written as:

$$\forall m_1 \in M, m_2 \in M, m_1 \neq m_2 :$$
$$s_{m_1} = s_{m_2} \vee t_{m_1} = t_{m_2} \rightarrow \neg overlap(m_1, m_2)$$

As discussed in section II-A, a long message may need to be broken into several pieces to fit into the bin. Those pieces must share the same path and be continuous in time. Let m' and m'' be two successive pieces of a message, then the following constraints must be enforced:

- the piece m'' appears one segment later than m' : $t(\frac{\mathbf{x}_{m'}}{w_{m'}}, \frac{W}{w_{m'}}) + 1 = t(\frac{\mathbf{x}_{m''}}{w_{m''}}, \frac{W}{w_{m''}})$,
- the offset within segment is 0 if it is not the first piece: $\mathbf{y}_{m''} = 0$,
- the same links are used: $\forall(i,j) \in B : \mathbf{k}_{m',i,j} = \mathbf{k}_{m'',i,j}$.

IV. Heuristic Approach

Section In most cases, message scheduling is only one part of the design process and needs to be carried out multiple times. However, as the problem size increases, the long execution time of a purely SMT based approach might become a hurdle for the designer. To cope with this problem, we propose an incremental heuristic to improve the scalability. The algorithm proceeds in three steps, detailed in the next section: 1) PE-to-switch allocation, 2) classical strip packing, 3) level packing. The general idea of the heuristic is to reuse the exist bin-packing algorithms to place the objects (step 2) and rely on the SMT solver to handle the non-standard constraints, i.e. overlapping of objects (step 3).

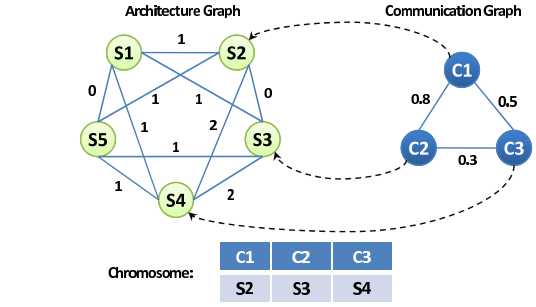

Fig. 5. PE-to-Switch Allocation Optimization

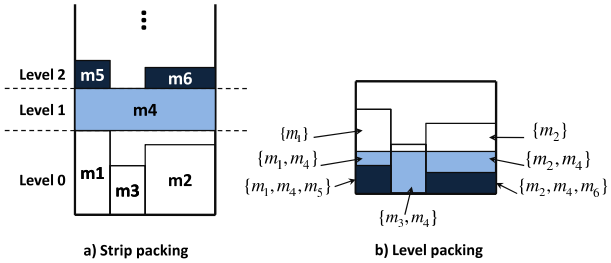

Fig. 6. Example of Strip Packing and Level Packing

1) PE-to-switch allocation. The goal of this step is to find a PE-to-switch allocation scheme π that minimizes the communication cost estimated by the distance between source and target nodes:

$$Minimize : cost = \sum_{m\in M} \frac{l_m}{p_m} * distance(\pi(s_m), \pi(t_m))$$

For that we adopt an Evolutionary Algorithm (EA) based optimization approach. The algorithm takes an architecture graph G_A and a communication graph G_C (Figure 5) as input. The architecture graph is a full-meshed graph with vertices corresponding to virtual components in the TTNoC. The weighted edges specify the distance in hops between virtual components. The communication graph is also a full-meshed graph with the vertices representing PEs. The edges describe the communication requirement between any two PEs, which is computed by:

$$R(C_1, C_2) = \sum_{m\in M\wedge((s_m=C_1\wedge t_m=C_2)\vee(s_m=C_2\wedge t_m=C_1))} \frac{l_m}{p_m}$$

A PE-to-switch allocation maps each vertex of G_C to one vertex of G_A. This mapping can be encoded as a list of integers as depicted in Figure 5. Standard operators such as two-point crossover can be adopted during the EA-based optimization process.

2) Strip Packing. This step packs the objects into an imaginary strip with infinite height as illustrated by Figure 6a. This problem has been extensively studied in the past (cf. [9]). Because of the good performance and simplicity in implementation, we adopt the First Fit Decreasing Height Decreasing Width (FFDHDW) algorithm in this paper. In

principle, any other existing algorithm could be used as well. The FFDHDW algorithm belongs to the category of *level algorithm*. These algorithms enforce the restriction that the objects are placed with the lower edge on certain horizontal levels [9]. The height of a level is determined by the height of the tallest object in that level. Using FFDHDW, the objects are pre-ordered in non-increasing height, and when height equals, non-increasing width. The objects are iteratively placed onto the lowest level with sufficient space and a new level is created on top of the current level if it does not fit any existing level. Recall that the objects transformed from the TTNoC messages are restricted to be placed into horizontal locations that are multiples of their width.

3) Level Packing. In this step, the levels are considered as one-dimensional objects with size equal to the height of the level and packed into bins (Figure 6b). As previously discussed, messages can overlap in time as long as a spatial separation is guaranteed. Thus, we try to overlay different levels to reduce the overall height of the strip. An outline of the level packing algorithm is presented in Algorithm 1. After strip packing, we iteratively place levels in decreasing height into the bin. The vector $locations$ contains the possible vertical locations to place the level. We try to place the level in the lowest possible location (line 4). If it is successful, the position and routing of the messages contained in this level will be computed and fixed later on. The top of the current level is considered as a possible location for future levels (line 5). This procedure is demonstrated in Figure 7. If a level fails at all locations, the messages are added to the *failedMessage* set and we move on to the next level (line 10).

Algorithm 1 IncrementalMessagePacking(): iterative level based bin-packing with intersection of objects allowed. The function $place$ uses SMT solver to check the feasibility. M: the set of messages.

```
1:  locations = {0};
2:  for all l ∈ levels with decreasing height do
3:    for all a ∈ locations in increasing order do
4:      if place(l,a)=success then
5:        locations = locations ∪ {a + height(l)}
6:        break;
7:      end if
8:    end for
9:    if l fails at all locations then
10:     addToFailedMessages(l)
11:   end if
12: end for
13: for all m ∈ failedMessages do
14:   place(m)
15: end for
```

The feasibility of placing a level at a certain location is checked by the SMT solver. Since packing and routing of existing messages in the bin are fixed, the corresponding SMT variables are replaced by constants in the constraints to simplify the formulation. The constraints to check if level

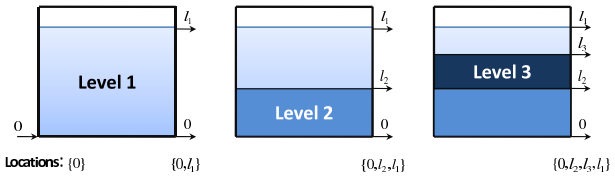

Fig. 7. Example of Level Packing

l can be placed in location x is:

$$\forall m \in l, m' \in existingMessages(x):$$
$$overlap(m,m') \rightarrow \bigwedge_{(i,j)\in B} (\mathbf{k}_{m',i,j} = 1 \rightarrow \mathbf{k}_{m,i,j} = 0)$$

The SMT solver can be granted the freedom to change the horizontal location of messages inside levels. In many cases, only certain combination of messages causes an unroutable case. It will be much more efficient to resolve these conflicts by moving the messages in the horizontal direction than by placing levels at different locations. For example, if messages m_5 and m_1 in Figure 6 cannot be routed together (e.g. they are from the same sending TISS), the area $\{m_1, m_4, m_5\}$ becomes unroutable and $level$ 2 cannot be placed at location 0. However, this can be resolved by moving m_5 to the right, e.g. to the same location as m_3. After placing all levels, an additional fixing step can be introduced that tries to allocate the set of failed messages (line 15 to 17 in Algorithm 1). We iteratively consider all messages and give the SMT solver the full freedom to change the location in both horizontal and vertical axis, i.e. messages are not restricted to any levels. Since only the variables associated with a single message need to be computed, such a problem is expected to be solved in a short time. Our experimental results verify this assumption.

V. Experiments

The SMT formulation and incremental heuristic are implemented in JAVA using the Z3 SMT solver [10]. The program is running on a Windows machine with 3GHz CPU and 4GB memory. We tested the scheduling algorithms on three architectures, namely a 3x3 TTNoC with 9 switching nodes, a 5x5 architecture with 25 nodes and a 7x7 architecture with 49 nodes. Three algorithms are compared:

- Pure SMT formulation (*SMT*).
- Incremental heuristic without fixing failed messages (*H-NoFix*).
- Incremental heuristic with the fixing step (*H-WithFix*).

We generate messages with periods between $32\mu s$ and $32ms$ and random length. For each architecture, a random set of 5 to 50 messages is generated and allocated. We execute 15 such test cases and compute the average results. In each round, the execution is terminated if the runtime exceeds 1.5 hour. Figure 8 compares the runtime of the three algorithms. The error bars depict the best and worst results obtained. As can be seen, the heuristic algorithm scales far better than a pure SMT solution. In case of 40 messages, a speedup of $58x$ and $30x$ is achieved by *H-NoFix* for the 3x3 and 5x5 TTNoC, respectively. The pure SMT solution exceeds the 1.5

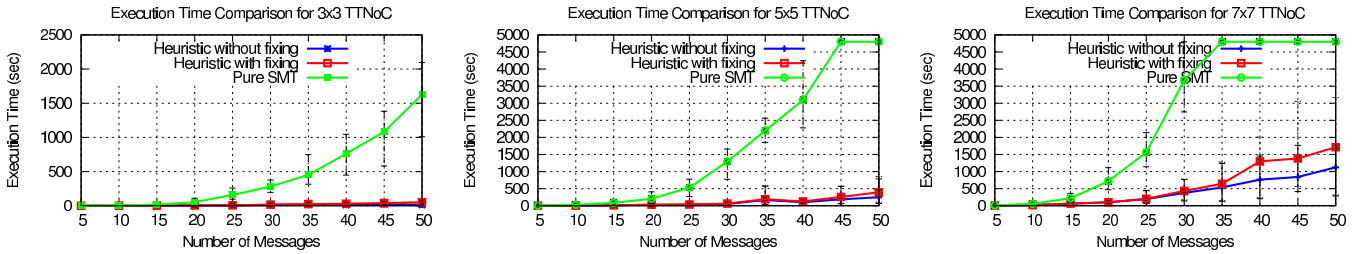

Fig. 8. Execution Time Comparison

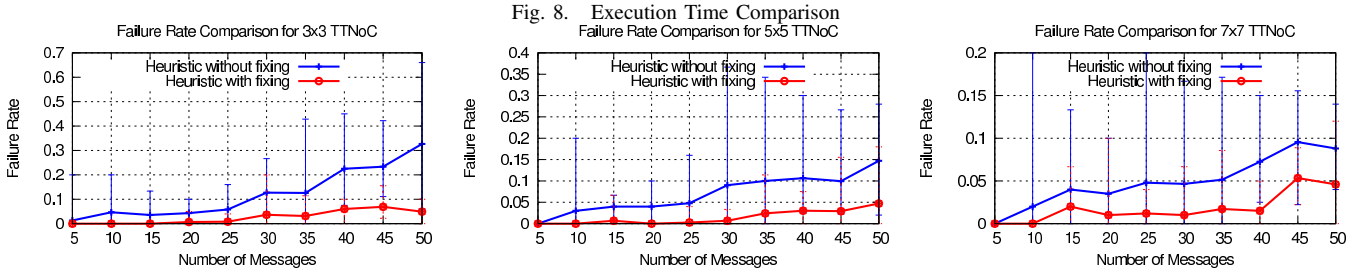

Fig. 9. Error Rate of Heuristic Algorithms

hour budget as the number of messages approaches 45 for the 5x5 architecture and 35 for the 7x7 architecture. The execution time of *H-WithFix* is very close to the *H-NoFix* algorithm. The extra time spent on fixing increases with the total number of messages. This is due to the fact that more messages fail as the problem becomes more difficult.

To evaluate the performance of the heuristic, we consider the percentage of messages failed to be allocated using the algorithm. Figure 9 presents the results. It can be seen that the failure rate generally decreases as the architecture size becomes larger. For the case of 50 messages, failure rates of 33%, 14% and 9% respectively are observed by *H-NoFix*. A very likely explanation is that mapping the same amount of messages to a larger architecture is generally a simpler problem due to more routing options. The *H-WithFix* algorithm further improves the performance. Again for the 50 message case, the failure rate is reduced to round 5% after the incremental fixing step. In Table I, we compare the number of successful/failed/expired test runs for some representative setups. A test case is considered as successful if all messages are allocated and failed if at least one message cannot be allocated. For relatively simple test cases (e.g. tests with less than 25 messages), the success rate of the heuristic (*H-WithFix*) is comparable with that of *SMT*. For larger tests (e.g. those with 50 messages), the success rate of heuristic is relatively low. Nevertheless, only a small portion of messages (less than %5) cannot be allocated. The designer may manually map the remaining messages or explore a larger architecture. As the counterpart, the SMT approach fails to provide a result due to expiration of time budget.

		SMT			Heuristic			
arch	num. mess.	#succ. case	#fail. case	#exp. case	#succ. case	#fail. case	#exp. case	#failed mess.(%)
3x3	25	13	2	0	13	2	0	0.8
5x5	25	15	0	0	14	1	0	0.3
7x7	25	15	0	0	12	3	0	1.2
3x3	50	13	2	0	1	14	0	4.9
5x5	50	0	1	14	3	12	0	4.7
7x7	50	0	1	14	2	13	0	4.6

TABLE I
COMPARING THE NUMBER OF SUCCESSFUL/FAILED/EXPIRED TESTS

VI. CONCLUSION

This paper tackles the TTNoC message scheduling problem. We transform the problem into a special case of 2D bin-packing and formulate it as an SMT instance. Since the scalability of a pure SMT based solution is very limited, we propose an incremental heuristic that combines SMT solving with a classical bin-packing algorithm. Experimental results show that a significant speedup is achieved with an acceptable performance loss (around 5% on average). As future work we plan to integrate heuristic routing algorithms into the bin-packing algorithm to further improve the scalability.

ACKNOWLEDGMENT

This work has been supported in part by the European research project ACROSS under the Grant Agreement ARTEMIS-2009-1-100208.

REFERENCES

[1] H. Shah, A. Raabe, and A. Knoll, "Bounding wcet of applications using sdram with priority based budget scheduling in mpsocs," in *DATE*, 2012.
[2] C. Paukovits, "The time-triggered system-on-chip architecture," Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Dec. 2008.
[3] H. Zeng, W. Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli, "Scheduling the flexray bus using optimization techniques," in *DAC*, 2009.
[4] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich, "FlexRay Schedule Optimization of the Static Segment," in *CODES+ISSS*, 2009.
[5] P. Milbredt, B. Vermeulen, G. Tabanoglu, and M. Lukasiewycz, "Switched FlexRay Increasing the Effective Bandwidth and Safety of FlexRay Networks," in *EFTA*, Bilbao, Spain, 2010.
[6] T. Schenkelaars, B. Vermeulen, and K. Goossens, "Optimal scheduling of switched flexray networks," in *DATE*, 2011.
[7] M. Lukasiewycz, S. Chakraborty, and P. Milbredt, "Flexray switch scheduling - a networking concept for electric vehicles," in *DATE*, 2011.
[8] W. Steiner, "An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks," in *RTSS*, 2010.
[9] N. Ntene and J. H. van Vuuren, "A survey and comparison of level heuristics for the 2d oriented strip packing problem," in *submitted to Applied Discrete Optimization*, 2006.
[10] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*, 2008.