Resource-efficient Routing and Scheduling of Time-constrained Network-on-Chip Communication*

Sander Stuijk, Twan Basten, Marc Geilen, Amir Hossein Ghamarian and Bart Theelen
Eindhoven University of Technology, Department of Electrical Engineering, Electronic Systems
s.stuijk@tue.nl

Abstract. Network-on-chip-based multiprocessor systems-on-chip are considered as future embedded systems platforms. One of the steps in mapping an application onto such a parallel platform involves scheduling the communication on the network-on-chip. This paper presents different scheduling strategies that minimize resource usage by exploiting all scheduling freedom offered by networks-on-chip. Our experiments show that resource-utilization is improved when compared to existing techniques.

1. Introduction

Increasing computational demands from multimedia applications have led to the development of multi-processor systems-on-chip (MP-SoC) which integrate many processing cores and memories. With the growing number of cores integrated into a chip, communication becomes a bottleneck as traditional communication architectures are inherently non-scalable [3]. Networks-on-Chip (NoC) are emerging as a communication architecture which solves this issue as it provides a better structure and modularity [3, 5, 16]. Furthermore, it can provide guarantees on the timing behavior of the communication. This enables the development of systems with a predictable timing behavior which is key for modern multimedia systems [7].

Current NoCs like Æthereal [16] and Nostrum [12] use circuit-switching to create connections through the NoC which offer timing guarantees. Today's routing and scheduling solutions however (a) often do not use all routing flexibility of NoCs and (b) make bandwidth reservations for connections with throughput/latency guarantees that are unnecessarily conservative. To illustrate the first point, for example, the scheduling strategies presented in [9, 11] restrict themselves to minimal length routes. Modern NoCs allow the use of other, more flexible, routing schemes. As an illustration of the second point, consider a simple NoC with three links l_1, l_2 and l_3. The data streams sent over l_1 and l_2, shown in Fig. 1, are both sent over l_3. Traditional NoC scheduling strategies [9, 10] reserve two guaranteed throughput connections on the link ($l_{3,trad}$). However, given the timing of the data streams on l_1 and l_2, it is possible to combine both streams and preserve bandwidth ($l_{3,new}$). The essential idea is not to reserve bandwidth for guaranteed throughput connections permanently during the entire execution of an active application but only during certain intervals. This paper explores and compares several new strategies which exploit all scheduling freedom offered by modern NoCs and minimize resource usage. Scheduling strategies which minimize resource usage will be able to schedule problems with tighter latency constraints and/or larger bandwidth requirements.

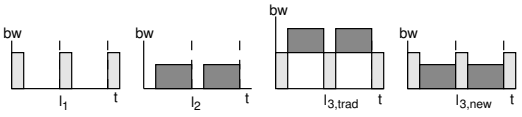


Figure 1. Motivating example.

The remainder of this paper is organized as follows. The next section discusses related work. Sec. 3 presents the application model used for programming NoC-based MP-SoC architectures. The architecture itself is discussed in Sec. 4. The time-constrained scheduling problem is formalized in Sec. 5. Several different scheduling strategies are presented in Sec. 6. The benchmark used to evaluate these strategies is presented in Sec. 7. The experimental results are discussed in Sec. 8.

2. Related Work

This paper considers scheduling streaming communication on a NoC within given timing constraints while minimizing resource usage. We restrict ourself to communication with timing constraints. In practice, some communication streams in an application may have no timing requirements. Scheduling techniques for these streams are studied in e.g. [15]. Those techniques can be used together with our approach to schedule both the communication without and with timing constraints.

In [11], a state-of-the-art technique is presented to schedule time-constrained communications on a NoC when assuming acyclic, non-streaming communication. That is, tasks communicate at most once with each other. Our application model, presented in Sec. 3, allows modeling of communication streams in which tasks periodically, i.e., repeatedly, communicate with each other.

Scheduling streaming communication with timing guarantees is also studied in [9, 10, 16]. They apply a greedy heuristic and reserve bandwidth for streams, whereas we propose to reserve bandwidth per message and present several different heuristics. Our results show a clear improve-

*This work was supported by the Dutch Science Foundation NWO, project 612.064.206, PROMES.

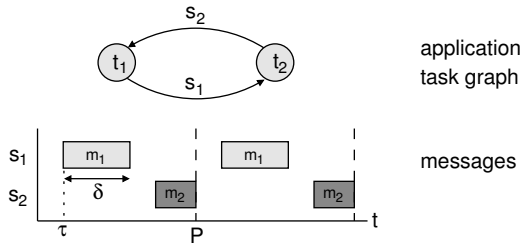


Figure 2. Example application task graph.

ment in resource usage. In [13], an extension to [10, 16] is presented to schedule multiple applications onto a single network-on-chip. This approach can also be used in conjunction with our scheduling techniques.

3. Application Modeling

Multimedia applications, for instance an MP3 encoder, operate on streams of data. These applications can be described by an application task graph in which the tasks are periodically executed. The period is determined by the throughput requirements of the application. Whenever a task executes, it exchanges messages with other tasks via (data) streams. Fig. 2 shows an example of a simple application task graph consisting of two tasks t_1 and t_2. Within each period P, task t_1 sends a message m_1 through stream s_1 to task t_2 and t_2 sends a message m_2 through s_2 to t_1. Note that the periodic execution of the tasks results in a periodic behavior of the communication between them, with potentially some data-dependent jitter.

To meet the computational requirements of modern multimedia applications, multi-processor systems are used. The tasks, from an application graph, are mapped to the various processors in the system. Whenever multiple tasks are mapped to one processor, the execution order of these tasks is fixed through a schedule. These schedules and the timing constraints imposed on the application determine time bounds within which each task must be executed. Similarly, they determine time bounds within which messages must be communicated between the tasks. This paper presents techniques to schedule messages on the NoC, which are specified with such time bounds.

4. Architecture Platform

Multiprocessor systems-on-chip, like Daytona [1], Eclipse [17], Hijdra [2], and StepNP [14], use the tile-based multiprocessor template described by Culler [4]. Each *tile* contains one or a few processor cores and local memories. The architecture template used in our work fits also in this template. A network-on-chip (NoC) is used to interconnect the different tiles. Each tile contains a *network interface* (NI) through which it is connected with a single router in the NoC. The *routers* are connected to each other in an arbitrary topology. The connections between routers and between routers and NIs are called *links*.

In this paper, the connections between the processing elements and the NI inside a tile are ignored. We assume that these connections introduce no delay, or that the delay is already taken into account in the timing constraints imposed on communications, and that there is sufficient bandwidth available. Hence, the NI can be abstracted away into the tile. Given this abstraction, the architecture can be described with the following graph structure.

Definition 1. (ARCHITECTURE GRAPH) *An architecture graph (N,L) is a directed graph where each node $u,v \in N$ represents either a tile or a router, and each edge $l = (u,v) \in L$ represents a link from node u to node v.*

Communication between tiles involves sending data over a sequence of links from the source to the destination tile. Such a sequence of links through the architecture graph is called a *route* and is defined formally as follows.

Definition 2. (ROUTE) *A route r between node u and node v with $u \neq v$ is a path in the architecture graph of consecutive links from u to v without cycles. The operators src and dst give respectively the source and destination node of a route or a link. The length of a route r is equal to the number of links in the path, and denoted $|r|$. We use $l \in r$ to denote that the link l appears in the route r.*

Links can be shared between different communications by using a TDMA-based scheduler in the routers and NIs. All links have the same number of TDMA *slots*, N, and each slot has the same bandwidth. At any moment in time, at most one communication can use a slot in a link. This guarantees that the NoC schedule is contention-free. Hence, no deadlock will occur. The data transfered over a link in a single slot is called a *flit* and it has size sz_{flit} (in bits). To minimize buffering in routers, a flit entering a router at timeslot t must leave the router at slot $t+1$. Not all slots in a link may be available for use by a single application. Part of the slots may already be used by other applications mapped to the system. The function $\angle : L \rightarrow 2^N$ associates with every link a set indicating which slots are free.

Wormhole routing [6] is used to send the flits through the network. This technique requires limited buffering resources and offers strict latency bounds. A message is divided by the sending NI into flits. The flits are then routed through the network in a pipelined fashion. This reduces the communication latency considerably. All flits which belong to the same message and are sent in consecutive slots form a *packet*. The first flit in a packet (header flit) contains all routing information and leads the packet through the network. The header has a fixed size of sz_{ph} bits ($sz_{ph} \leq sz_{flit}$). The remaining $sz_{flit} - sz_{ph}$ bits in the header flit can be used to send (a part of) the actual message. The size of the header must be taken into account when allocating resources in the NoC. Two messages, possibly sent between different source

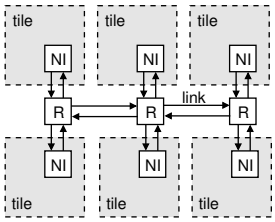


Figure 3. NoC-based MP-SoC architecture.

and destination tiles but over one link at non-overlapping moments in time can use the same slot. For messages that use the same slot in the link between the source tile and the first router but a different route, the routing information stored in the NI for this slot must be changed. This can be implemented efficiently by sending a message from a processor or communication assist [4] inside a tile to its NI to change the routing information. The time required to reconfigure the NI is T_{reconf}. During this reconfiguration time, the slot may not be used to send messages.

Tasks in an application communicate with each other through streams of messages. The ordering of the messages in a stream must be preserved. To realize this, the NIs send messages onto the network in the same order as they receive them from the processors. The scheduling of communications on the NoC must also guarantee that the messages are received in the same order. No reordering buffers are thus needed in the NIs, which simplifies their hardware design. The NoC further requires that when the communication of a message is started, slots are claimed in the links it is using. These slots are only freed after the communication has ended. Preemption of a communication is not supported.

5. Time-Constrained Scheduling Problem

Informally, this paper tries to find a schedule for a set of messages which are sent, within given timing constraints, between different tiles in a system. A message is formally defined as follows.

Definition 3. (MESSAGE) *Given an architecture graph (N,L), a set of streams S and a period P. A message m is a 7-tuple (u,v,s,n,τ,δ,sz), where $u,v \in N$ are respectively the source and the destination tile of the n-th message sent through the stream $s \in S$ during the period P. The earliest time at which the communication can start, relative to the start of the period, is given by $\tau \in \mathbb{N}$ $(0 \leq \tau < P)$. The maximum duration of the communication after the earliest start time is $\delta \in \mathbb{N}$ $(\delta \leq P)$. The size (in bits) of the message that must be communicated is $sz \in \mathbb{N}$.*

In the application task graph shown in Fig. 2, a message $m_1 = (u,v,s_1,n,\tau,\delta,sz)$ is sent each period through the stream s_1. This communication can start at time τ and must finish before $\tau+\delta$. Note that a communication may start in some period and finish in the next period. This occurs when $\tau+\delta > P$.

In practice, messages may not always have a fixed earliest start time, duration, or size. Conservative estimates on these figures should be used to construct the set of messages in order to guarantee that all communications fall within the timing and size constraints. Resources that are claimed but not used, due to for example a smaller message size, can be used to send data without timing requirements between tiles without providing guarantees, i.e. best effort traffic.

A message specifies timing constraints on the communication of data between a given source and destination tile. It does not specify the (actual) start time, duration, route and the slot allocation. This information is provided by the *scheduling entity*.

Definition 4. (SCHEDULING ENTITY) *A scheduling entity is a 4-tuple (t,d,r,st), where $t \in \mathbb{N}$ is the start time of the scheduled message relative to the start of the period and $d \in \mathbb{N}$ is the duration of the communication on a single link. The scheduled message uses the route r in the network and the slots it uses from the slot table of the first link $l \in r$ are contained in the set $st \in 2^N$.*

The slots given in st are claimed on the first link of the route r at time t for the duration d. On the next link, the slot reservations are cyclically shifted over one position. So, these slots are claimed one time-unit later, i.e. at $t+1$, but for the same duration d. The complete message is received by the destination at time $t+d+|r|-1$. Fig. 4 shows a scheduling entity which sends a message over a link with a slot-table of 8 slots. Starting at time $t=2$, the slots 2, 3, and 4 are used to send the message. The communication ends after $d=11$ time units. In total two packets consisting both of three flits are used to send the message.

The relation between a message and a scheduling entity is given by the *schedule function*, formally defined below in Def. 5. Among all schedule functions, those respecting the constraints in Def. 5 are called *feasible*. The first two constraints make sure that the communication takes place between the correct source and destination tile. The third and fourth constraint guarantee that the communication falls within the timing constraints given by the message. The fifth constraint ensures that enough slots are reserved to send the message and packet headers over the network. It uses a function $\pi(e)$ which gives for a scheduling entity $e=(t,d,r,st)$ the number of packets which are sent between t and $t+d$ on the first link of the route r considering the slot reservations st and assuming that at time 0 the first slot of the slot table is active. The function $\varphi(e)$ gives the number of slots reserved by e between t and $t+d$. The sixth constraint makes sure that a scheduling entity does not use slots in links which are used by other applications. It uses a function $\sigma(e,l_k) = \{(s+k) \bmod N \mid s \in st\}$ which gives for a scheduling entity e and the k-th link l_k on the route r of e the set of slots it uses from the slot-table with size N. The sev-

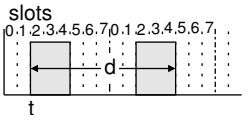


Figure 4. Scheduling entity on a link.

enth constraint requires that the schedule is contention-free. The next constraint makes sure that there is enough time to reconfigure the NI between two messages which originate at the same NI and use the same slot but different routes. The last constraint enforces that the ordering of messages in a stream is preserved.

Definition 5. (SCHEDULE FUNCTION) *A schedule function is a function $\mathcal{S} : M \rightarrow E$ where M and E are respectively the set of messages and scheduling entities. We call $\mathcal{S}$ feasible if and only if, for all messages $m = (u, v, s, n, \tau, \delta, sz) \in M$ associated to scheduling entity $\mathcal{S}(m) = e = (t, d, r, st)$,*

1. *the route starts from the source tile: $u = src(r)$,*
2. *the route ends at the destination tile: $v = dst(r)$,*
3. *the communication does not start before the earliest moment in time at which the data is available: $t \geq \tau$,*
4. *the communication finishes not later than the deadline: $t + d + |r| - 1 \leq \tau + \delta$,*
5. *the number of allocated slots is sufficient to send the data: $sz + sz_{ph} \cdot \pi(e) \leq sz_{flit} \cdot \varphi(e)$,*
6. *for all $l \in r$, $\sigma(e, l) \cap \mathcal{L}(l) = \emptyset$,*

and for each pair of messages $m_1, m_2 \in M$ with $m_1 \neq m_2$, $m_1 = (u_1, v_1, s_1, n_1, \tau_1, \delta_1, sz_1)$, $\mathcal{S}(m_1) = e_1 = (t_1, d_1, r_1, st_1)$, $m_2 = (u_2, v_2, s_2, n_2, \tau_2, \delta_2, sz_2)$, and $\mathcal{S}(m_2) = e_2 = (t_2, d_2, r_2, st_2)$,

7. *for all $l \in r_1 \cap r_2$, with l the i-th link in r_1 and l the j-th link in r_2 and $[t_1 + i, t_1 + d_1 + i] \cap [t_2 + j, t_2 + d + j] \neq \emptyset$ must hold: $\sigma(e_1, l) \cap \sigma(e_2, l) = \emptyset$,*
8. *if $u_1 = u_2$, $r_1 \neq r_2$, and $st_1 \bigcap st_2 \neq \emptyset$, then there is enough time to reconfigure the NIs: $(P + t_2 \bmod P - (t_1 + d_1) \bmod P) \bmod P \geq T_{reconf}$,*
9. *if $s_1 = s_2$ and $n_1 < n_2$, then the ordering of these messages is preserved: $t_1 + d_1 < t_2 \wedge t_1 + d_1 + |r_1| - 1 < t_2 + |r_2|$.*

If a schedule function is not feasible, it means that one or more of the above rules are violated in at least one associated scheduling entity. Such a schedule is called *infeasible*. By construction, any feasible schedule is contention-free and hence free of deadlock and livelock [8].

6. Scheduling Strategies

6.1. Overview

Given a set of messages M, a scheduling strategy must find a schedule entity e for each message $m \in M$ and the set of scheduling entities E must form a feasible schedule function (i.e. all constraints from Def. 5 must be met). Given that an exhaustive approach is not tractable, we present several heuristic approaches. The heuristics allow the user to trade off quality of solutions and effort spent on solving problems. First, a greedy strategy is presented in Sec. 6.2. Typically, the greedy approach gives a solution quickly. However, it also excludes a large part of the solution-space. The second strategy, ripup, adds backtracking to the greedy approach. This improves the quality (number of feasible solutions found for a set of problems), but it also increases the run-time. The backtracking tries to resolve scheduling conflicts when they occur. The third strategy, presented in Sec. 6.4, tries to avoid conflicts by estimating a priori the usage of all links. This should steer the routing process to avoid scheduling conflicts and as such minimizes the use of the backtracking mechanism.

6.2. Greedy

The greedy strategy explores a small part of the solution-space. As a result, it has a small run-time. However, it may miss solutions or find non-optimal ones in terms of resource usage. The greedy strategy essentially tries to schedule the largest, most time-constrained messages first, via the shortest, least congested route that is available. It works as follows. First, all messages $m \in M$ are assigned a cost using Eqn. 1 and sorted from high to low based on their cost. The cost function guarantees that messages are ordered according to their (integer) size (larger size first) and that two messages with the same size are ordered with respect to the duration (tighter constraint first).

$$\text{cost}_M(m) = sz(m) + \frac{1}{\delta(m)} \tag{1}$$

Next, a schedule entity $e = (t, d, r, st)$ must be constructed for the first message $m = (u, v, s, n, \tau, \delta, sz) \in M$. To minimize the resource usage, the scheduling strategy must try to minimize the length of the routes. For this reason, the greedy strategy determines a list R of all routes from u to v with the shortest length and assigns a cost to each route r using the following cost function that determines the minimum ratio of free slots versus available slots in a route.

$$\text{cost}_R(r, m) = \min_{l \in r} \frac{\mathcal{F}(l, m)}{|\mathcal{L}(l)|}, \tag{2}$$

with $\mathcal{F}(l, m)$ the number of slots in the link l that are not used between $\tau(m)$ and $\tau(m) + \delta(m)$ by the set of schedule entities E constructed so far and with $\mathcal{L}(l)$ the number of slots available in the link l. The routes are sorted from high to low cost giving preference to the least congested routes. Next, a schedule entity e is constructed using the first route r in R. The scheduling strategy should avoid sending data

in bursts as this increases the chance of congestion. Therefore, the start time, t, of e is set equal to the earliest possible time respecting the third and last constraint from Def. 5. Given t and the fourth and last constraint from Def. 5, the maximal duration d of e can be computed. All slots available between t and the maximal duration on the first link of the route, respecting the sixth, seventh and eighth constraint from Def. 5, are located. From these slots, a set of slots, st, is selected which offer sufficient room to send the message and the packet headers. The scheduler tries to minimize the number of packets that are used by allocating consecutive slots in the slot table. This minimizes the overhead of the packet headers, which in turn minimizes the number of slots needed to sent the message and its headers. This leaves as many slots as possible free for other messages. It is possible that no set of slots can be found which offer enough room to send the message within the timing constraints. If this is the case, the next route in R must be tried. In the situation that all routes are unsuccessfully tried, a new set of routes with a length of the minimum length plus one is created and tried. This avoids using routes longer than needed and it never considers a route twice. A route which uses more links than the minimum required is said to make a *detour*. The length of the detour is equal to the length of the route minus the minimum length. If no set of slots is found when a user-specified maximum detour of X is reached, then the problem is considered infeasible. If a set st of slots is found, the minimal duration d needed to send the message via the route r, starting at time t using the slots st is computed using the fifth constraint from Def. 5. The scheduling entity $e = (t, d, r, st)$ is added to the set of schedule entities E. The new set of schedule entities $E \cup \{e\}$ is guaranteed to respect all constraints from Def. 5. The next message can be handled. The process is repeated till a schedule entity is found for all messages in M, or until the problem is considered infeasible (a message cannot be scheduled).

6.3. Ripup

The ripup strategy uses the greedy strategy described in the previous section to schedule all messages. This guarantees that all problems that are feasible for the greedy strategy are also solved in this strategy. Moreover, the same schedule function is found. As soon as a conflict occurs (i.e. no schedule entity e_i can be found for a message m_i which meets the constraints given in Def. 5), an existing schedule entity e_j is removed from the set of schedule entities E. To choose a suitable e_j, the heuristic calculates for each schedule entity $e_j \in E$ the number of slots it uses in the links that can also be used by e_i. The higher this number, the larger the chance that e_j forms a hard conflict with e_i. A schedule entity e_j with the largest conflict is therefore removed from E. This process is continued until a schedule entity e_i for the message m_i can be created that respects the constraints given in Def. 5. After that, the messages of which the corresponding schedule entities were removed are re-scheduled in last-out first-in order. On a new conflict, the ripup mechanism is activated again. The user specifies the maximum number of times a ripup may be performed. This allows a trade-off between quality and run-time of the strategy.

6.4. Global knowledge

The ripup scheduler does not know a priori which unscheduled messages need to use links in the route it assigns to the message it is scheduling. It can only use local information to avoid congestion. The *global knowledge* strategy tries to estimate, before scheduling messages, the number of slots that are needed in each of the links. This gives the scheduling strategy global knowledge on the congestion of links. This knowledge is used to guide the route selection process when scheduling the messages.

Communication of a message m can take place at any moment in time within the time interval specified by m. Within this interval it requires at least $\left\lceil \frac{\lceil sz(m)/sz_{flit} \rceil}{\max(\lfloor \delta(m)/N \rfloor, 1)} \right\rceil$ slots in each link of the route it uses. In the optimal situation, all scheduled messages use a route with the shortest length. To estimate the congestion on all links in the NoC, the strategy assumes that only shortest length routes are used. For each link $l \in L$, the strategy computes the minimal number of slots required at each moment in time when all messages which can use l, as it is part of at least one of their shortest routes, would use the link l. The function $C : L \times \mathbb{N} \rightarrow \mathbb{N}$ gives the estimated number of slots used in a link $l \in L$ at a given time t.

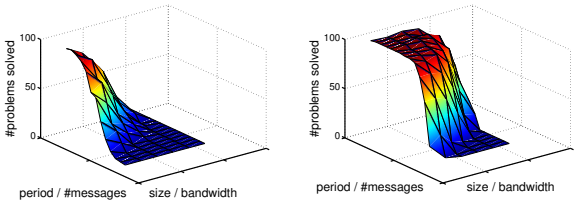
The global knowledge strategy uses the same algorithm as the ripup strategy. However, a different cost function is used to sort the routes it is considering when scheduling a message. The cost function used by the greedy and ripup strategy (Eqn. 2) is replaced by the following cost function.

$$\text{cost}_R(r, m) = \sum_{l \in r} \max_{\tau(m) \leq t < \tau(m) + \delta(m)} C(l, t) \qquad (3)$$

This cost function ensures that the routes are sorted based on the estimated congestion of the links contained in the routes. Routes containing only links with a low estimated congestion are preferred over routes with links that have a high estimated congestion. This minimizes the number of congestion problems which occur during scheduling. As such, it makes more effective use of the allowed ripups.

6.5. Cost functions

Cost functions are used in the scheduling strategies to sort the messages M and routes R. The cost functions should minimize the chance of having a conflict when scheduling messages. They are constructed in such a way that the most resource constrained messages are handled first and that the



(a) 5x5 mesh with greedy. (b) 5x5 mesh with knowledge.

Figure 5. Feasible problems in the problem-space.

resource usage is balanced over all links in the NoC. However, by doing so, they up-front exclude points from the solution-space. To circumvent this problem, randomly ordered sets M and R can be used as an alternative for the cost functions. In our experiments, we do so to study the impact of the cost functions on the quality of the strategies.

7. Benchmark

A benchmark is needed to test the quality of the scheduling strategies. It must contain a set of problems that covers a large part of the problem space typical of realistic problems. It should also be large enough to avoid optimization towards a small set of problems. Currently, no benchmark is available which meets these requirements. It is not possible to construct a benchmark containing only real existing applications. Profiling these is too time-consuming and they are not representative for more demanding future applications at which NoCs are targeted. Therefore, we chose to create a benchmark which consists of a set of randomly generated problems.

As in [9, 10, 11], we use a mesh topology in our evaluation. Tiles located at the edge of the mesh are restricted in the links that can be used as at least one direction is not available because of the topology. In a 3x3 mesh this holds for all tiles except for one. In a 5x5 mesh there are 16 edge tiles and 9 non-edge tiles and a 7x7 mesh has 24 edge tiles and 25 non-edge tiles. The ratio of edge to non-edge tiles can possibly influence the scheduling strategies. To study this effect, problem sets are generated for a 3x3, 5x5, and 7x7 mesh.

A traffic generator is developed which creates a user-specified number of streams of messages between randomly selected source and destination tiles. The streams can model uniform and hotspot traffic. All messages in a stream are assigned a start time, size, and duration which consists of a randomly selected base value which is equal for all messages in the stream plus a random value selected for each individual message in the stream. The first part can be used to steer the variation in message properties between streams. The second part can be used to create variation between messages in a single stream (i.e. jitter).

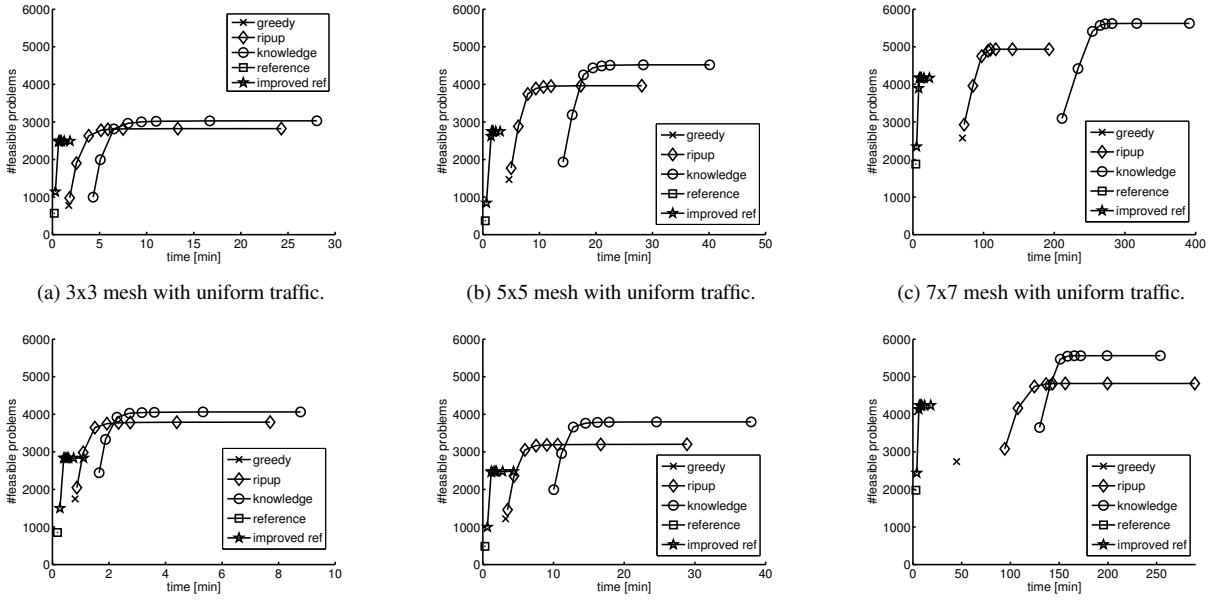
The problem space can be characterized in a 2-dimensional space. The first dimension is determined by the number of messages which must be communicated within a period. The second dimension is determined by the ratio of the size of the messages communicated and the available bandwidth. When constructing the problem sets, we found that there is an area in the problem-space where problems change from being easy to solve to unsolvable. We selected 78 equally distributed points around this area in the problem-space. For each point we generated 100 problems. This gives a benchmark with in total a set of 7800 different problems per mesh-size and traffic model. Fig. 5 shows for each point in the problem-space of the 5x5 mesh with uniform traffic how many problems are solved with the greedy and global knowledge strategies. The results for the greedy strategy show that most problems do not have a trivial solution. A solution is found for only 19% of the problems. The results of the global knowledge strategy show that 59% of the problems can be solved (and already suggest that global knowledge performs better than greedy). So, our benchmark contains problems which are not trivial to solve (i.e. greedy does not find a solution), but a solution does exist (i.e. global knowledge finds a solution). Note that when the problems get harder to solve, the demands on the resources are increased. More latency sensitive messages and/or larger messages (more bandwidth) need to be scheduled. Scheduling strategies which are more resource efficient will be able to solve more problems.

8. Experimental Results

8.1. Reference scheduling strategies

A state-of-the-art scheduling strategy is presented in [10]. The strategy allows the use of non-shortest routes but it assumes that slots cannot be shared between different streams. Reconfiguration of the NIs is not possible. As in our greedy strategy, this strategy does not reconsider scheduling decisions when a conflict occurs. We used this strategy in our experiments as our *reference* strategy. It is implemented using the greedy strategy with three restrictions imposed on it. One, messages in one stream must use the same route. Two, streams are not allowed to share slots. Three, the reconfiguration time is equal to a period. This makes it impossible to reconfigure the NIs. The experimental results suggest that using a backtracking mechanism is very effective. For this reason, we extended the reference strategy with our ripup mechanism. This strategy is used in the experiments as the *improved reference* strategy.

We also replaced the cost functions in the ripup strategy with a mechanism which assigns random costs to messages and routes. Experiments showed that the number of times we run this strategy with a fixed number of ripups on a given problem set did not have an influence on the number of problems for which a feasible schedule function is found. On the other hand, increasing the number of ripups led to an increase in the number of solved problems. However, the



(a) 3x3 mesh with uniform traffic.

(b) 5x5 mesh with uniform traffic.

(c) 7x7 mesh with uniform traffic.

(d) 3x3 mesh with hotspot traffic.

(e) 5x5 mesh with hotspot traffic.

(f) 7x7 mesh with hotspot traffic.

Figure 6. Trade-off between feasible problems and run-time.

number of problems solved within a limited run-time and randomly ordered messages and routes is far lower than the number of problems solved by any of the heuristics in the same time. This shows that the cost functions in the heuristics are effective in ordering the messages and routes. For brevity, we do not discuss these experiments in more detail.

8.2. Experiments on the benchmark

All scheduling strategies have been tested on the benchmark problems. The ripup, global knowledge and improved reference strategies have been tested with a number of different values for the maximum number of ripups (1, 10, 50, 100, 150, 200, 400, 800) to study the trade-off between the number of problems for which a solution is found and the run-time. A slot-table size of 8 slots is used in all experiments and the maximum detour (X) is initially set to 0. Note that $X = 0$ guarantees that any solution uses only shortest routes. This allows us to study for how many problems each strategy is able to find a solution with minimal resource requirements. The reconfiguration time of the NI, T_{reconf}, is set to 32 time units. This gives tiles 4 complete rotations of the slot table to reconfigure the NI. A processor or communication assist must send a message to update the routing information in the NI. The size of this message is less than the size of a single flit (i.e. it needs one time unit to be sent), so the value for T_{reconf} is conservative.

The trade-off between the run-time and the number of problems that is solved with the various strategies is shown in Fig. 6. Tab. 1 summarizes these results for all strategies assuming that 800 ripups are allowed. The column 'Improvement' shows the percentage of additional problems that is solved by all strategies compared to the reference strategy. The column 'Avg time' gives for each strategy the average run-time on a problem.

Looking at the number of problems solved, the results show that the reference strategy is outperformed by the improved reference strategy. This shows that adding backtracking to the state-of-the-art scheduling algorithm presented in [10] improves the results considerably. The results show further that the reference strategy solves less problems than greedy and the improved reference strategy solves less problems than the other two strategies using ripups. From this, we conclude that not using the ability of NoCs to reconfigure their connections is a limiting factor. As modern NoCs do not have this limitation, problems scheduled using the reference strategies may unnecessarily be considered infeasible or use unnecessarily many resources. Slot sharing is especially advantageous for hotspot traffic. For this type of traffic, our strategies are able to solve up-to 65% more problems than the improved reference strategy. This shows that slot sharing reduces the problem of contention on links connected to a hotspot.

The results show also that the global knowledge strategy always outperforms the other strategies. However, the average run-time on a problem is larger for this strategy than for the other strategies. This is caused by the congestion estimation made at the start of the strategy. Simpler estimates might be used to reduce its run-time. The reference and improved reference strategy have always the lowest run-time. This is logical as route selection is done only once for all messages in a stream and the slot allocation does not have to consider reconfiguration of slots.

Table 1. Experimental results.

	Improvement	Avg time [ms]	Detour ($X = 2$)
Greedy	47%	161	110%
Ripup	248%	731	31%
Knowledge	334%	974	17%
Reference	0%	11	4%
Improved ref.	209%	67	16%

Modern NoCs allow the use of flexible routing schemes (i.e. routes may use a detour). More problems may be solved when this flexibility is used. To quantify this gain, we tested all strategies with a maximum detour of 2 on all problems in our benchmark. The results of this experiment are shown in column 'Detour' of Tab. 1, which shows the improvement in the number of problems solved when compared to the same strategy with detour zero. It shows that using non-shortest routes helps in solving additional problems.

8.3. Experiments on a multimedia system

Besides the synthetic streams, a realistic multimedia application consisting of an MPEG-4 decoder and an MP3 decoder is used in the experiments. All 15 tasks in the application task graph are mapped and scheduled manually onto a 2x2 mesh and through profiling a set of 16 streams describing the decoding of a video frame with accompanying audio is derived. For each scheduling strategy, the minimal slot table size is determined for which a feasible schedule is found. When only shortest routes are used, both the reference strategies, including the existing method, require a slot table with 6 slots. Our strategies require a slot table with 2 slots. This shows that for a realistic application slot sharing reduces the resource requirements on the NoC (i.e. fewer slots need to be allocated for the application). When the maximum detour is 2 links, the reference strategies require a slot table with 4 slots and our strategies require a slot table with 1 slot. This confirms that using non-minimal routes reduces the requirements on the NoC.

9. Conclusion

This paper studies the problem of scheduling time-constrained communication of a streaming application on a NoC. Several new strategies are presented to route and schedule streaming communication. The scheduling strategies use all routing and scheduling flexibility offered by modern NoCs while limiting resource usage. Short routes and the reservation of consecutive slots in slot tables minimize resource usage and packetization overhead. However, they also create potential bottlenecks in the NoC, which may render some resources unusable for scheduling other streams. The use of non-minimal routes and non-consecutive slot reservations might increase scheduling freedom for remaining streams. Our strategies try to find a good compromise in the allocation of routes and slot-table slots. The experiments show that our strategies perform better than the state-of-the-art strategy of [10]. The reason is that our strategies exploit freedom offered by modern NoCs not used in the existing strategy. Additionally, we found that adding backtracking to this state-of-the-art strategy improves its results considerably with only a small overhead on its run-time.

References

[1] B. Ackland, et al. A single chip 1.6 billion 16-b MAC/s multiprocessor DSP. *IEEE Journal of Solid-State Circuits*, 35(3):412–424, 2000.

[2] M. Bekooij, et al. Predictable multiprocessor system design. In *SCOPES'04, Proc.*, pages 77–91. Springer, 2004.

[3] L. Benini and G. de Micheli. Networks on chips: A new SoC paradigm. *IEEE Comp.*, 35(1):70–78, 2002.

[4] D. Culler, et al. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[5] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *DAC'01, Proc.*, pages 684–689. ACM, 2001.

[6] W. J. Dally and C. L. Seitz. The torus routing chip. *Journal of distributed computing*, 1(3):187–196, 1986.

[7] O. Gangwal, et al. *Dynamic and Robust Streaming In and Between Connected Consumer-Electronics Devices*, chapter Building Predictable Systems on Chip: An Analysis of Guaranteed Communication in the AEthereal Network on Chip, pages 1–36. Springer, 2005.

[8] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Computer Architecture, Proc.*, pages 278–287, 1992.

[9] K. Goossens, et al. A design flow for application-specific networks on chip with guaranteed performance to accelerate SoC design and verification. In *DATE'05, Proc.*, pages 1182–1187. IEEE, 2005.

[10] A. Hansson, et al. A unified approach to constrained mapping and routing on network-on-chip architectures. In *CODES+ISSS*, pages 75–80. IEEE, 2005.

[11] J. Hu and R. Marculescu. Communication and task scheduling of application-specific networks-on-chip. *IEE Proceedings: Computers and Digital Techniques*, 152(5):643–651, 2005.

[12] M. Millberg, et al. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *DATE'04, Proc.*, pages 890–895. IEEE.

[13] S. Murali, et al. A methodology for mapping multiple use-cases onto networks on chip. In *DATE'06, Proc.*, pages 118–123. IEEE, 2006.

[14] P. Paulin, et al. Application of a multi-processor SoC platform to high-speed packet forwarding. In *DATE'04, Proc.*, pages 58–63. IEEE, 2004.

[15] E. Rijpkema, et al. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEE Proceedings: Computers and Digital Techniques*, 150(5):294–302, 2003.

[16] A. Rădulescu, et al. An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration. *IEEE Trans. on CAD of ICs and systems*, 24(1):4–17, 2005.

[17] M. Rutten, et al. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Design & Test of Computers*, 19(4):39–50, 2002.