TU/e technische universiteit eindhoven

Time-Constrained Energy-Aware Routing and Scheduling of Network-on-Chip Communication

Sander Stuijk, Amir Hossein Ghamarian, Twan Basten, Marc Geilen, Bart Theelen

ES Reports
ISSN 1574-9517

ESR-2005-08
14 July 2005

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Time-Constrained Energy-Aware Routing and Scheduling of Network-on-Chip Communication

Sander Stuijk, Amir Hossein Ghamarian, Twan Basten, Marc Geilen 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 (NoC-based MP-SoCs) are considered as future embedded systems platforms. One of the steps in mapping an application onto such a parallel platform involves determining when data is sent between the tasks in the application. As a network-on-chip allows concurrent communication, a decision on the route through the network must also be made. Timing constraints and energy consumption are aspects that must be considered in this mapping. This paper presents different routing and scheduling strategies which try to minimize the resource usage and energy consumption when an application with timing constraints is mapped onto a NoC-based MP-SoC. Our experiments show that our strategies outperform an existing state-of-the-art scheduling technique, and that in some cases random-based strategies perform the best.

1 Introduction

Consumers have high expectations about the quality delivered by next-generation multimedia applications. For example, they expect that their new mobile MP3-player produces an audio stream without hick-ups and that the battery lifetime increases with each generation. Systems must guarantee that applications produce their results within strict time bounds. They should also try to minimize their energy consumption. Multiprocessor systems-on-chip (MP-SoCs) are increasingly used in multimedia systems as their energy consumption is typically lower than that of a single-processor solution. They satisfy the increasing demand for compute power by combining many existing IP-blocks and memories. To connect these blocks, networks-on-chip (NoC) are often mentioned as they provide a structured mechanism for the interconnection.

Applications that must be mapped onto an MP-SoC are often described using task graphs. Timing constraints, which must be met during the mapping, are imposed on the tasks in the task graph. The timing constraints represent the required throughput or maximal latency of a task. These timing constraints impose time bounds between which data must be sent between the tasks. Resources must be reserved in the NoC to communicate the data within the time bounds. A route must be selected through the NoC and a schedule for the data sent over the route must be made. As energy consumption is an important design criterion, the scheduling strategy should try to minimize the energy consumption. In this paper, we introduce and compare different routing and scheduling strategies for data which is sent within strict timing constraint over a NoC. State-of-the-art scheduling strategies [HM05] do not exploit all freedom offered by current NoCs. Our experiments show that by exploiting this freedom it is possible to find routes and schedules for a larger set of mapping problems. Our experiments show also that for modestly sized NoCs a random-based strategy outperforms all heuristics.

The remainder of this paper is organized as follows. The next section discusses related work on scheduling and routing for NoCs and traditional computer-networks. The NoC-based MP-SoC architecture is presented in Sec. 3. The time-constrained scheduling problem is formalized in Sec. 4.1. In Sec.

4.2, this problem is proved to be NP-complete. Sec. 5 continues with a discussion on energy optimization for NoCs. Four different energy-aware scheduling strategies are presented in Sec. 6. The construction of the benchmark used to evaluate these strategies is presented in Sec. 7. The experimental results are discussed in Sec. 8.

2 Related work

Routing and scheduling has been researched before in the area of direct networks for parallel and distributed systems. A routing technique which has been developed in this field and which is also interesting for NoCs is wormhole routing [DS86]. It requires limited buffering resources and offers strict latency bounds. These are two important aspects in the field of NoCs. Therefore, we believe this to be the most appropriate routing technique for NoCs. Examples of NoCs with these properties are Æthereal [Rad05] and Nostrum [Mil04]. A comprehensive overview of wormhole techniques in direct networks is presented in [NM93]. An important objective in NoC routing is to find deadlock-free routes. Several deadlock-free adaptive routing algorithms have been proposed in the field of direct networks [Chi00, GN92]. In NoCs, the focus on reducing the energy consumption and the stricter latency and throughput requirements force us to rethink the techniques developed in this field and adapt them to the context of NoCs.

Scheduling of data communication has also been studied before in the field of NoC-based MP-SoCs. In [HM05], Hu and Marculescu present a heuristic to perform a combined mapping of tasks to processors and the mapping of edges to routes in the NoC. As in our work, their objective is to meet the timing constraints while minimizing the energy consumption. They assume that a mesh topology is used with routers that support an XY-routing scheme. Furthermore, they assume that the bandwidth in a link cannot be shared among different communications. Current NoCs allow bandwidth sharing and the use of other routing schemes. So, using Hu and Marculescu's approach, a problem may be considered infeasible while a solution exists if a NoC with the afore mentioned freedom is used. In the current paper, we explore several strategies exploiting all routing and scheduling freedom. We compare these strategies, using a mesh topology, with Hu and Marculescu's approach as a reference point.

3 Architecture platform

Multiprocessor systems-on-chip, like Daytona [Ack00], Eclipse [Rut02], Hijdra [Bek04], and StepNP [Pau04], use the tile-based multiprocessor template described by Culler [Cul99]. 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 is used to interconnect the different tiles. Each tile contains a *router* to which other tiles can be connected. Fig. 1 shows an instance of our architecture template in which the tiles are connected using a mesh topology. Our approach supports also other topologies (e.g. tree or torus). The connections between routers are called *links*. Each link has a *bandwidth* which expresses the maximal amount of data it can transfer per time unit. The bandwidth that is available in the link can be shared among different communications. The routers use wormhole-based routing [DS86]. In this paper, the connections between the processing elements and the router 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 router can be abstracted away into the tile. A tile is thus directly connected to other tiles through links. This abstraction conforms to the models used by Hu et al. in [HM05]. Given this abstraction, the architecture can be described with the following graph structure.

Definition 1. (ARCHITECTURE GRAPH) *An architecture graph (T,L) is a directed graph where each vertex $t \in T$ represents a tile, and each edge $l_k = (t_u, t_v) \in L$ represents a link from tile t_u to tile t_v. Self edges, i.e. edges from a tile to itself, are not allowed.*

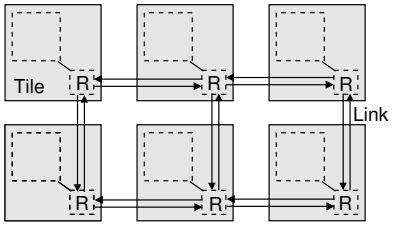


Figure 1: NoC-based MP-SoC architecture.

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 $r_{u,v}$ between tile t_u and tile t_v with $t_u \neq t_v$ is a sequence of links $l_m, l_{m+1}, \ldots, l_n$ of the architecture graph. The operators src and dst give respectively the source and destination tile of a route or a link. For a route $r_{u,v}$ holds that:*

- *the source of the first link $src(l_m)$ is equal to the source tile of the route $src(r_{u,v})$ and the destination of the last link $dst(l_n)$ is the destination tile $dst(r_{u,v})$.*
- *for any two consecutive links l_k, l_{k+1} in a sequence $dst(l_k) = src(l_{k+1})$;*
- *there is no cycle in a route, i.e. for any two links $l_k \neq l_l$ in the sequence holds $dst(l_k) \neq dst(l_l)$;*

The length of a route $r_{u,v}$ is equal to the number of links in its sequence, and denoted with $|r_{u,v}|$. We use $l_m \in r_{u,v}$ to denote that the link l_m appears in the route $r_{u,v}$.

The bandwidth of a link, which can be shared among different communications, is divided into a fixed number of equally-sized *slots*. The function $\mathcal{L} : L \rightarrow \mathbb{N}$ gives the number of slots available in a link. Not all links offer necessarily the same number of slots. This is useful in the context of routing and scheduling communications on a NoC onto which already other applications are mapped (i.e. not all slots in a link may still be available). Each slot in each link offers the same bandwidth (bw_{slot}). This can, for example, be implemented using a TDMA-based scheduler in the routers [Bek04]. Two data-elements, possibly sent between different source and destination tiles but over one link at non-overlapping moments in time can use the same slot. This can be realized by sending the routing information along with each data-element. We assume that a data-element cannot be subdivided into smaller parts which can be communicated independently. As soon as the communication of a data-element is started, it claims slots in the links it is using. These slots are only freed after the communication has ended. So, pre-preemption of communication is not supported. This is a logical assumption as in NoCs a data-element is sent at once to the router in the tile.

4 Time-constrained scheduling problem

4.1 Problem formulation

Informally, this paper tries to find a schedule for a set of data-elements that must be communicated, within given time bounds, between different tiles in a system. These data-elements with their time bounds are referred to as *communication events* and are defined as follows.

Definition 3. (COMMUNICATION EVENT) *Given an architecture graph (T, L). A communication event c is a 5-tuple $(u, v, \tau_s, \tau_e, sz)$, where $u, v \in T$ are respectively the source and the destination tile of a communication event. The earliest time at which the communication can start is given by $\tau_s \in \mathbb{N}$. The latest time at which the communication may end is given by $\tau_e \in \mathbb{N}$ ($\tau_e > \tau_s$). The size (in bits) of the data-element that must be communicated is $sz \in \mathbb{N}$.*

A communication event specifies time bounds within which a data-element must be sent between a given source and destination tile. It does not specify the (actual) start and end times, the route and the amount of bandwidth that must be allocated. This information is provided by the *scheduling entity*.

Definition 4. (SCHEDULING ENTITY) *A scheduling entity is a 4-tuple (t_s, t_e, r, s), where $t_s \in \mathbb{N}$ and $t_e \in \mathbb{N}$ are respectively the start time and the end time of the scheduled communication event. The scheduled communication event uses the route r in the network and on each link $l \in r$ it uses $s \in \mathbb{N}$ slots.*

A scheduling entity provides a time frame within which a communication event can be sent along the route r. The used bandwidth is given by the number of slots (s) reserved in the link multiplied with the bandwidth of a single slot (bw_{slot}). Fig. 2 shows an example of a communication event that is scheduled on a link l. The communication is performed within the time bounds (τ_s and τ_e) and it uses two slots in the link. These slots are claimed between t_s and t_e; outside this time interval the slots can be used to communicate other data-elements. A scheduling entity with one slot is also able to transfer the data-element within the time bounds.
The relation between a communication event and a scheduling entity is given by the *schedule function*, formally defined below. Among all schedule functions, those respecting the constraints in Def. 5 are called *valid*. 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 actual start and end time of the communication fall within the bounds given by the communication event. The fifth constraint says that there should be enough time between the start and end time of the communication to sent all data along the whole route. Routing information must be sent with each data-element. The amount of time needed to set-up a router to sent a data-element along the correct route is $t_{router\ setup} \in \mathbb{N}$. This time is also taken into account in the fifth constraint. The last constraint guarantees that the number of slots used in any link at any moment in time never exceeds the number of slots available in the link. The function α, used in this constraint, gives the number of slots used in a link l by a scheduling entity e at time t[1]. The number of slots is zero if the link is not used in the route r given by e or the time t is outside the start and end time given by e.

Definition 5. (SCHEDULE FUNCTION) *A schedule function is a function $\mathcal{S} : C \rightarrow E$ where C and E are respectively the set of communication events and scheduling entities. If $c = (u, v, \tau_s, \tau_d, sz)$ is a communication event and $\mathcal{S}(c)$ associates it to the scheduling entity $e = (t_s, t_e, r, s)$, then we call $\mathcal{S}$ valid if and only if for all $c \in C$,*

- *the route starts from the source tile: $u = src(r)$,*
- *the route ends at the destination tile: $v = dst(r)$,*
- *the communication does not start before the earliest moment in time at which the data is available: $t_s \geq \tau_s$,*
- *the communication finishes in time: $t_e \leq \tau_e$,*
- *the bandwidth given by the scheduling entity e is sufficient to send the data: $t_e \geq t_s + \frac{sz}{s \cdot bw_{slot}} + t_{router\ setup} \cdot (|r| + 1)$,*

and for the set of communication events C holds:

$$\forall t \in \mathbb{N} \wedge l \in L \sum_{c \in C} \alpha(\mathcal{S}(c), l, t) \leq \mathcal{L}(l),$$

with

$$\alpha(e, l, t) = \begin{cases} s(e) & \text{if } t_s(e) \leq t \leq t_e(e) \wedge l \in r(e); \\ 0 & \text{else.} \end{cases}$$

If a schedule function is not valid, it means that one or more of the above rules are violated in at least one associated scheduling entity. A schedule function with at least one violating scheduling entity is called *infeasible*.

[1] For a tuple $z = (x, y, \ldots)$, we use $x(z)$ to denote x of z, and $y(z)$ for y, etcetera.

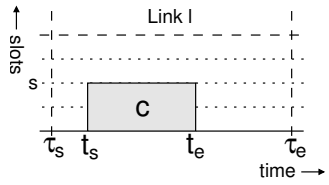


Figure 2: A scheduled communication event.

4.2 Complexity

In this section, we prove that the scheduling problem is NP-complete.

Theorem 1. *Given an architecture graph $G(T,L)$ and a set of communication events C. The problem of finding a valid schedule function which associates C to a set of scheduling entities E is NP-complete.*

We first show that the problem belongs to NP. The verification algorithm must check whether the communication events satisfy the validity constraints of Def. 5. The first five constraints can be checked in $O(|C|)$ time. The last constraint can also be checked in $O(|C|)$ time for a single link $l \in L$, but it has to be repeated for all $l \in L$. So, the complexity of the verification algorithm is $O(|L||C|)$, which is polynomial in the problem size. To prove that the problem is NP-complete, we show that the disjoint-path problem [GJ79] can be reduced in polynomial time to our scheduling problem. The disjoint-path problem was proved to be NP-complete (even for planar graphs) [GJ79]. In the disjoint-path problem, a set of edge-disjoint paths in a given graph must be found between a set of pairs of vertices. The reduction of the disjoint-path problem to our scheduling problem works as follows. Let G be the graph and the set $\{(s_1,d_1)\dots(s_k,d_k)\}$ the pairs of vertices that form an instance of the disjoint-path problem. We construct an instance of the scheduling problem with architecture graph G and the number of slots in each link equal to 1. The set of communication events $C = \{(s_1,d_1,0,1,1),\dots(s_k,d_k,0,1,1)\}$, i.e., all events have size 1, deadline 1 and no further timing constraints. This construction can be done in polynomial time. Suppose that there are edge-disjoint paths $p_1,\cdots,p_k$ between $(s_1,d_1)\dots(s_k,d_k)$. For each i, the path p_i, which is in fact a route in the scheduling problem, is exclusively dedicated to the communication event $(s_i,d_i,0,1,1)$. The event is scheduled to be sent at time 0 using all bandwidth in the links on its route. It is easy to see that this schedule function is valid. Conversely, suppose that there is a valid schedule function for the set C; then, the set of scheduling entities cannot share bandwidth, as sharing bandwidth leads to missing deadlines. Since bandwidths are all set to be 1, it trivially follows that the communication routes are all disjoint. Hence, any valid schedule is a solution to the disjoint-path problem.

5 Energy optimization problem

Energy modeling for networks-on-chip has been studied in [HM05, Ye,03]. For NoCs with buffers implemented by registers, both Hu et al. in [HM05] and Ye et al. in [Ye,03] suggest taking the sum of the energy consumed in the router (E_R) and on the link (E_L) as the energy needed to transfer one bit. In [HM05], Eqn. 1 is given to compute the average energy consumption for sending one bit of data from tile u to tile v using the route $r_{u,v}$.

$$E_{bit}^{u,v} = |r_{u,v}| \cdot (E_R + E_L) + E_R \tag{1}$$

E_R is constant and also E_L can be considered constant. The latter can be achieved if all tiles are of (more or less) equal size. Note that this still allows heterogeneity in the tiles. A communication event c consists of a set of bits ($sz(c)$) that must be sent from tile u to tile v. The used route is given by the valid schedule function $\mathcal{S}$. Using Eqn. 1, we find that the energy used for sending c is equal to:

$$\begin{aligned} E_c &= sz(c) \cdot E_{bit}^{u,v} \\ &= sz(c) \cdot |r(\mathcal{S}(c))| \cdot (E_R + E_L) + sz(c) \cdot E_R \end{aligned}$$

The energy consumption of a set of communication events C is equal to the sum of the energy consumed by all $c \in C$.

$$E_C = \sum_{c \in C} sz(c) \cdot |r(\mathcal{S}(c))| \cdot (E_R + E_L) + sz(c) \cdot E_R \tag{2}$$

In this paper, we try to find scheduling strategies which give a valid schedule and minimize the energy consumption (i.e. minimize the value of Eqn. 2). Since, both E_R and E_L are constants and the size of a communication event is fixed, minimization of the energy requires the minimization of the following cost function:

$$\text{cost}_E(C) = \sum_{c \in C} sz(c) \cdot |r(\mathcal{S}(c))| \tag{3}$$

The cost function shows that in order to minimize the energy used in the NoC, the weighted length of the routes used to sent data between the tiles should be minimized. The optimum is reached if all communication events use a route which is equal to a shortest path in the architecture graph. However, there may not always be a valid schedule function which uses only shortest routes.

6 Energy-aware scheduling strategies

6.1 Overview

Given a set of communication events C, the scheduling strategies must find a schedule entity e for each communication event $c \in C$ and the set of scheduling entities E must form a valid schedule function (i.e. all constraints from Def. 5 must be met) minimizing the cost function of Eqn. 3. Exhaustively searching the solution-space is impossible as the problem is NP-complete and the problem instances are large. The run-time for scheduling communication events on a 5x5 mesh is in the order of weeks. Therefore, algorithms that search only part of the space must be used. In the experimental evaluation of Sec. 8, we compare various strategies both on quality (number of feasible solutions found, energy minimization) and run-time. Acceptable run-times vary per application of the scheduling strategy (e.g. in design-time mapping trajectories or in run-time (re-)configuration or QoS managers).
First, a random-based strategy is presented in Sec. 6.2. It does up-front not exclude any point from the solution-space. It serves as an interesting reference point for the other strategies. The approach allows the user to specify parameters that trade-off run-time and effort spent on solving problems. As a second strategy, a greedy approach is presented in Sec. 6.3. Typically, a greedy approach gives a solution quickly. However, it also excludes a large part of the solution-space. Hence, it may miss solutions or find non-optimal ones. The third strategy, ripup, adds backtracking to the greedy approach. This will improve the results, but it will also increase the run-time. Also the ripup strategy allows a trade-off between run-time and quality. The last strategy, presented in Sec. 6.5, tries to combine the useful properties of the random-based and the greedy strategy. It adds to this a mechanism to get a global notion on which links are overloaded. Global knowledge is hard to capture in pure random or greedy strategies. The run-time is expected to increase compared to a greedy strategy, but it is expected that more problems are solved.

6.2 Random-based

The random-based strategy does not up-front exclude points from the solution-space and can therefore be used to get a reference to compare our scheduling strategies. The user influences the search-time by specifying the number of attempts that are performed to find a valid schedule function. For each attempt, all communication events C are placed in a random order. Next, a schedule entity $e = (t_s, t_e, r_{u,v}, s) \in E$ must be constructed for the first communication event $c = (u, v, \tau_s, \tau_e, sz) \in C$. The construction starts with placing all routes r between the tiles u and v with a length of at most the minimum length plus X ($|r| \leq min(|r_{u,v}|) + X$) in a random order in a list R. X, specified by the user, restricts the detour that a route may take. Typically, schedule entities should use short routes as this reduces the usage of the links in the NoC. However, sometimes a detour must be made to find a feasible solution. Because of the random ordering of the routes, a large X will lead to the situation in which often large routes are used.

This increases the chance of conflicts (i.e. not finding a feasible solution). For the first route $r_{u,v} \in R$, a random start time $t_s \geq \tau_s$ and end time $t_e \leq \tau_e$ are chosen. Using the fifth constraint from Def. 5, the minimum number of slots s needed is calculated. This gives a complete schedule entity e that respects the first five constraints of Def. 5. Next, it is checked whether the last constraint of Def. 5 is still met for all communication events in C for which so far a schedule entity in E has been constructed. If the constraint is not met, e is removed from E and an exhaustive search to find the first point in time to schedule c on the route $r_{u,v}$ is performed. Often, there are many points in time where c can be scheduled. The search will find such a point and avoid that the problem is considered infeasible because of one unfortunate random choice for a start or end time. If no entity that gives a valid schedule function is found, the next route from R is tried. This process is continued till either all routes in R are tried or the constraint is met. In the situation that the constraint is met, a schedule entity is constructed for the next communication event in C. This process is continued till all communication events are handled. In case all routes are tried without success, the attempt is considered unsuccessful and possibly a new attempt is performed. Such a new attempt starts from scratch by randomly regenerating the ordered sets C and R. If the maximum number of attempts is reached and no valid schedule function is found, the problem is considered infeasible.

6.3 Greedy

The greedy strategy excludes a large part of the solution-space. As a result, it typically gives a solution quickly. However, it may miss solutions or find non-optimal ones in terms of energy. The scheduling strategy works as follows. First, all communication events $c \in C$ are assigned a cost using Eqn. 4 and sorted from high to low based on their cost. The cost function guarantees that communication events are ordered according to their size (larger size first) and that two communication events with the same size are ordered wrt the time bounds (tighter bounds first).

$$\text{cost}_C(c) = sz(c) + \frac{1}{\tau_e(c) - \tau_s(c)} \tag{4}$$

Next, a schedule entity $e = (t_s, t_e, r_{u,v}, s)$ must be constructed for the first communication event $c = (u, v, \tau_s, \tau_e, sz) \in C$. The scheduling strategy should avoid sending data in bursts as this increases the chance of congestion. Therefore, the start and end time in e equal the time bounds of the corresponding c (i.e. $t_s = \tau_s$ and $t_e = \tau_e$). To optimize the energy consumption of the schedule, Eqn. 3 shows that 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 with the shortest length and assigns a cost to each route $r_{u,v}$ using the following cost function that determines the minimum ratio of free slots versus available slots in a route.

$$\text{cost}_R(r_{u,v}, c) = \min_{l \in r_{u,v}} \frac{\mathcal{F}(l,c)}{\mathcal{L}(l)}, \tag{5}$$

with $\mathcal{F}(l,c)$ the number of slots in the link l that are not used between $t_s(c)$ and $t_e(c)$ by the set of schedule entities E constructed so far. 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 $r_{u,v} \in R$ and the minimal number of slots s found using the fifth constraint from Def. 5. This schedule entity e is added to the set of schedule entities E. The new set of schedule entities $E \cup \{e\}$ must still respect the last constraint from Def. 5. If this is the case, the next communication event can be handled. Else, e must be removed from E and 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. If still no schedule entity is found such that the last constraint of Def. 5 is met when a user-specified maximum detour of X is reached, then the problem is considered infeasible. The process is repeated till a schedule entity is found for all communication events in C, or until the problem is considered infeasible.

6.4 Ripup

The ripup strategy uses the greedy strategy described in the previous section to schedule all communication events. 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 constructed for a communication event c_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 times it appears in the links that can also be used by e_i. The more often a schedule entity e_j appears, the larger the chance is that it forms a real 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 communication event c_i can be created that respects the constraints given in Def. 5. After that, the communication events 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.

6.5 Random with global knowledge

The experiments, discussed in detail in Sec. 8, show that the random-based strategy outperforms the two heuristics for modestly sized NoCs. They also show that backtracking requires a larger than expected run-time. However, some choices made in the heuristic increase the speed at which a valid schedule function is found. Based on these observations, a strategy is developed which combines the best of the strategies presented so far. In addition, a global view of link overheads is computed in case of infeasibility of a schedule function. This global knowledge is not easily captured in a pure random or greedy approach. It can however be useful in deciding how to solve scheduling conflicts.

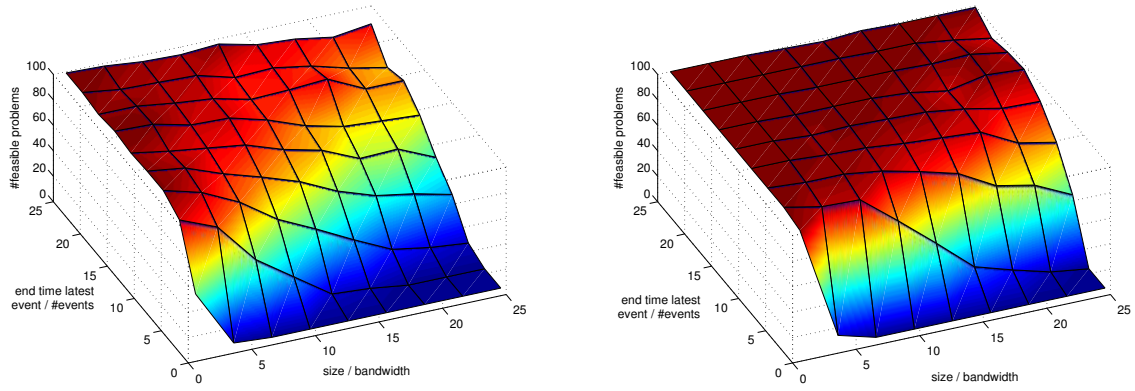
The strategy start with generating a schedule function $\mathcal{S}$. It constructs a schedule entity $e = (t_s, t_e, r_{u,v}, s) \in E$ for each communication event $c = (u, v, \tau_s, \tau_e, sz) \in C$. As in the random approach, each e uses a randomly chosen route $r_{u,v}$ from the set of routes R with minimal length. The start and end time are respectively $t_s = \tau_s$ and $t_e = \tau_e$, as in ripup, which avoids sending data in bursts. The first five constraints from Def. 5 are met using this construction technique. It must be checked whether also the last constraint is met. If not valid, then a ripup process is initiated using global knowledge. For each communication event c for which the last constraint is not met, a penalty is computed using Eqn. 6. The violation function $\mathcal{V}$ gives the number of slots that are needed on a link l at a time instance t but that are not available.

$$\mathcal{P}(c) = \sum_{l \in r_{u,v}} \sum_{t_s \leq t \leq t_e} \mathcal{V}(l,t), \tag{6}$$

with $r_{u,v} = r(\mathcal{S}(c))$, $t_s = t_s(\mathcal{S}(c))$, $t_e = t_e(\mathcal{S}(c))$, and

$$\mathcal{V}(l,t) = \max((\sum_{c_i \in C} \alpha(\mathcal{S}(c_i), l, t)) - \mathcal{L}(l), 0).$$

The communication events are sorted according to their penalty. Next, the schedule entity e_i of a communication event c_i with the largest penalty is removed from the set of schedule entities E. It is tried to construct a new schedule entity e_j for c_i of which the penalty is zero. This is done by performing an exhaustive search on all routes of at most minimum length plus a user-specified detour X for the earliest moment in time at which c_i can be scheduled. The routes are tried in the order of their length. If no e_j is found, the next communication event is removed from E and a schedule entity is searched for it. This process continues till all communication events with a non-zero penalty are handled. For each communication event c_i for which no new schedule entity e_j was found, the original e_i is inserted again in the set E. Next, the penalties are calculated again and it is tried to find a new schedule entity for each communication event with a non-zero penalty. This process continues till either the number of communication events with a penalty does not decrease or all communication events have a penalty of zero. In the latter case, a valid schedule function is found. In the first case, the problem is considered infeasible.



(a) 3x3 mesh with greedy.

(b) 3x3 mesh with random.

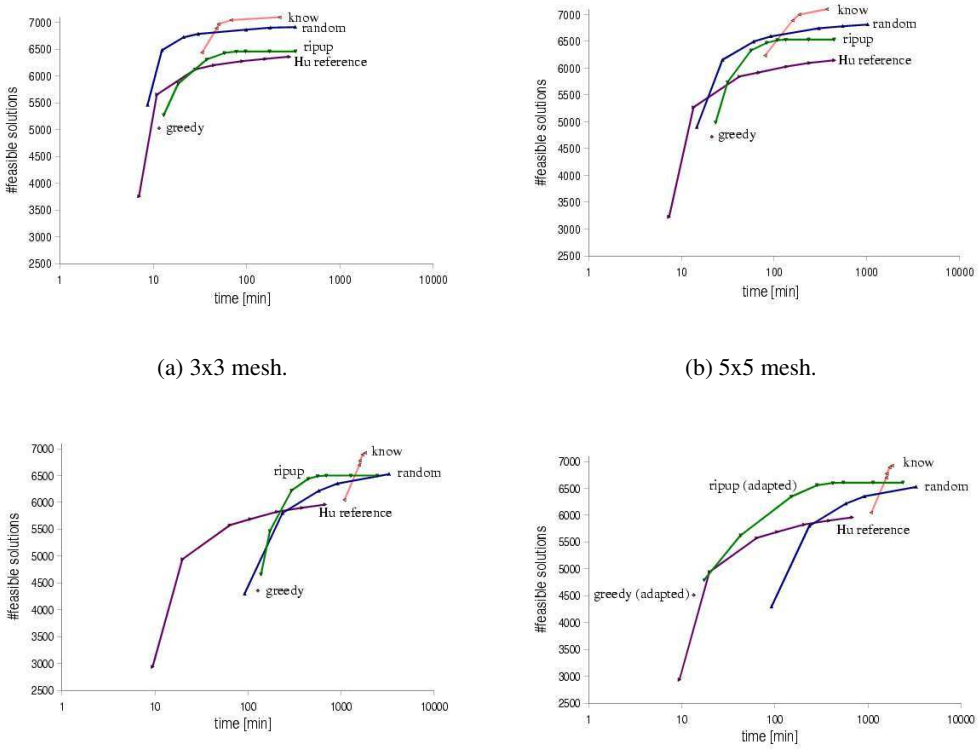
Figure 3: Feasible problems in the problem-space.

7 Benchmark

A benchmark is needed to test the quality of the scheduling strategies. This benchmark must contain a set of problems that covers a large part of the problem space given by existing, 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. Realistic applications cannot be used for this benchmark. It is too time-consuming to profile the communication behavior of a large number of applications. Furthermore, the communication behavior of applications can be very diverse which makes it hard to construct one representative set. Considering these problems, we chose to create a set of randomly generated problems. As in [HM05], 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. As a result, communication events that use these tiles are restricted by the topology in the possible routes. 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.

Given a topology in the form of an architecture graph, an instance of the scheduling problem can be characterized by five different factors. First, the size of the data-elements that must be communicated. Second, the bandwidth that is available in the links. Third, the number of communication events that must be scheduled. Fourth, the time within which all data-elements must be sent. Fifth, the time between the start and end time bound in the communication events. The first two factors can be captured in one ratio, as increasing the bandwidth will have the same effect on the feasibility of problems as decreasing the size of the data-elements. The third and fourth factor can also be captured in one ratio. The time within which all data-elements must be sent and the number of communication events in the problem have an inverse effect on the feasibility of the problems. We assume that each problem can be characterized by these two ratios. The fifth factor is ignored. This is reasonable if the number of problems is large enough, as all variants are then contained in the problems (i.e. the problem space is covered). Observe that decreasing one or both ratios should decrease the number of feasible problems.

A problem-space can be characterized in a 2-dimensional space with the above mentioned ratios on the axes. 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 81 equally distributed points around this area in the problem-space. For each point we generated 100 problems. This gives



(a) 3x3 mesh.

(b) 5x5 mesh.

(c) 7x7mesh.

(d) 7x7mesh with adapted greedy and ripup.

Figure 4: Trade-off feasible problems and run-time.

a benchmark with in total a set of 8100 different problems per mesh-size. Fig. 3 shows for each point in the problem-space of the 3x3 mesh how many problems are solved with the greedy and random-based strategy. The results for the greedy strategy show that there is a good mixture between simple and difficult problems. The results of the random-based strategy show that most problems can be solved (and already suggests that random performs better than greedy). So, our benchmark contains the right problems as they are not trivial to solve (i.e. greedy does not find a solution), but a solution does exists (i.e. random finds a solution). For each problem holds that it is never infeasible because of constraints set on a single communication event. However, it is unknown whether a problem is feasible. Furthermore, no energy-optimal solution is known for feasible problems. Consequently, it is only possible to make a relative comparison between the various scheduling strategies.

8 Experimental results

8.1 Hu and Marculescu's reference scheduling strategy

A state-of-the-art routing and scheduling strategy is presented in [HM05]. The strategy uses an XY-routing scheme and it assumes that bandwidth cannot be shared between communications. The objective of the strategy is to minimize the energy consumption while meeting the timing constraints. The energy consumption of the NoC is calculated using Eqn. 1, making the optimization objective identical to ours. This strategy is used in the experiments as our reference strategy. Following [HM05], it is implemented using the random-based strategy with two restrictions on it. One, only routes following the XY-routing scheme may be used. Two, the maximum bandwidth available in the links must be used. This makes it impossible to share the bandwidth in a link between schedule entities as the whole bandwidth in the link

is always occupied by a single schedule entity.

8.2 Comparison of scheduling strategies

All scheduling strategies have been tested on the benchmark problems. The tests were conducted on a Pentium III 1GHz with 4GB of internal memory. The random-based strategy has been tested with a number of different values for the maximum number of attempts (1, 10, 50, 100, 500, 1000, 2000). In all these experiments, the maximum detour (X) is set to 0. Our experiments showed that larger values of X increased the chance of choosing an infeasible solution. Note that $X = 0$ guarantees that the energy consumption of the solutions is optimal (i.e. Eqn. 3 is minimal) because only shortest routes are chosen. The benchmark has also been tested on the other strategy. In the ripup and the random with global knowledge strategies, the maximum number of ripups influences the run-time. This number has been varied (1, 10, 50, 100, 150, 200, 400, 800, and 1, 5, 10, 50, 100 respectively) to study the trade-off between the number of problems for which a solution is found and the run-time. The run-time of the greedy strategy cannot be influenced as in the other strategies. The maximum detour (X) is set in the greedy, ripup and random with global knowledge strategies to 2. Our experiments showed that this setting gives the best results.

Fig. 4(a)-(c) show the trade-off between the run-time and the number of problems that is solved with the various strategies. Looking at the number of problems solved, the results show that the reference strategy is outperformed by all other strategies except the greedy approach. From this, we conclude that using only the XY-routing scheme and disallowing communications to share bandwidth is too limited. As modern NoCs do not have these limitations, problems scheduled using this reference strategy may unnecessarily be considered infeasible. The results show also that for moderately sized NoCs the random-based strategies outperform the heuristics.
Fig. 4(c) shows that for larger meshes (7x7) the reference strategy is considerably faster than all our strategies. The reason for this is that in a mesh the number of shortest routes between two tiles grows exponentially in the mesh size, whereas in the XY-routing scheme of Hu and Marculescu there are at most two shortest routes. Since our strategies evaluate all routes of a certain length, they are topology independent, the evaluation takes more time, but often more solutions are found. It is known, however, that XY routes usually give quite good results in a mesh when communication events are uniformly distributed over the tiles. Our strategies can easily be adapted to take this property of meshes into account. Fig. 4(d) shows the results when the greedy and ripup strategy first try routes that follow the XY-routing scheme. They outperform the reference strategy both in the run-time and the number of problems solved. The experimental results show that when a large NoC (e.g. 7x7 mesh) is targeted a combination of random and heuristics should be used. A random-based strategy should be used when an application uses only a small part of the NoC (e.g. 3x3 tiles from a 7x7 mesh) and a heuristic should be used when all tiles in the NoC are used.
As a final remark on run-times, observe that the run-time for evaluating the full benchmark are sometimes large, but that the run-time of the strategies on individual problems is small. This may give options to perform routing and scheduling at run-time. To give a final answer whether this is possible further experiments on realistic applications and platforms are needed.

The objective of the strategies is to find a feasible solution which minimizes the energy consumption. The cost function of Eqn. 3 must therefore be minimized. If all used routes are of the shortest length, the energy cost is minimal. So, the solutions found with the random-based strategy or Hu and Marculescu's strategy are optimal as they only use routes with the minimal length. For each mesh size, the cost function of Eqn. 3 is computed for all problems of which both the random-based (Hu and Marculescu's) and ripup strategy can find a valid schedule function. Tab. 1 shows the percentage of additional energy that is needed for the solutions found with the ripup strategy compared to the solutions found with the random-based and Hu and Marculesu's strategy. It shows that the solutions found by the ripup strategy are typically close to the optimal solution. For the greedy strategy, we found that the solutions are even closer to the energy-optimal solutions. The comparison between the random with global knowledge on

Table 1: Additional energy usage ripup.

	vs Hu and Marculescu	vs Random
3x3 mesh	0.17%	0.25%
5x5 mesh	0.07%	0.09%
7x7 mesh	0.05%	0.06%

the one hand and the random-based or Hu and Marculescu's strategy on the other hand shows that the random with global knowledge strategy never requires 0.01% more energy than the optimal solution. So, also this strategy gives solutions close to the energy-optimal solutions.

9 Conclusions

In this paper, we formalized the problem of routing and scheduling a set of data-elements within strict timing constraints in a network-on-chip while minimizing the energy consumption. We proved that this problem is NP-complete. It is impossible to search the solution space exhaustively for reasonably sized problems. Four new strategies are presented to route and schedule the data-elements. Two strategies are based on a random algorithm; the other two are heuristics. A randomized approach potentially allows coverage of the whole solution space, whereas a heuristic up-front excludes points. A completely randomized algorithm was included as a reference to test the quality of the developed heuristics. The two heuristics essentially are a greedy approach and a greedy approach with backtracking. The final strategy combines the best elements of the random approach and the two heuristics. Our experiments show that for moderately sized NoCs, the random-based strategies outperform the heuristics. They show also that our strategies perform better than the state-of-the-art strategy of [HM05]. The reason is that our strategies exploit freedom offered by modern NoCs not used in the existing strategy.
The presented scheduling strategies assume that tasks have already been mapped and scheduled onto the tiles. It is an important step in a systems-on-chip design flow. In our future work, we want to consider this mapping and scheduling step and study the phase coupling with the routing and scheduling problem presented in this paper.

References

[Ack00] 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.

[Bek04] M. Bekooij, et al. Predictable multiprocessor system design. In *SCOPES'04, Proc.*, pages 77–91. Springer, 2004.

[Chi00] G. Chiu. The odd-even turn model for adaptive routing. *IEEE Transactions on parallel distributed systems*, 11(7):729–738, July 2000.

[Cul99] D.E. Culler, et al. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[DS86] W. J. Dally and C. L. Seitz. The torus routing chip. *Journal of distributed computing*, 1(3):187–196, 1986.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and interactability: a guide to the theory of NP-completeness*. W.H. Freeman and Co., 1979.

[GN92] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Int. symp. computer architecture, Proc.*, pages 278–287, May 1992.

[HM05] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 24(4):551–562, April 2005.

[Mil04] 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, 2004.

[NM93] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–76, 1993.

[Pau04] P.G. Paulin, et al. Application of a multi-processor SoC platform to high-speed packet forwarding. In *DATE'04, Proc.*, pages 58–63. IEEE, 2004.

[Rad05] A. Radulescu, et al. An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 24(1):4–17, January 2005.

[Rut02] M. Rutten, et al. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Design & Test of Computers*, 19(4):39–50, 2002.

[Ye,03] T.T. Ye, et al. Packetized on-chip interconnect communication analysis for MP-SoC. In *DATE'03, Proc.*, pages 344–349. IEEE, March 2003.