Scheduling in a Real-time Network-on-Chip

Period minimization using metaheuristics

Mark Ruvald Pedersen Jaspur Højgaard Rasmus Bo Sørensen

Technical University of Denmark, Kgs. Lyngby

{s072095, s072069, s072080}@student.dtu.dk

Abstract

This paper shows how two metaheuristics can be applied to optimize scheduling of of inter-processor communication in a real-time Network-on-Chip. This scheduling problem is an NP-complete multi-commodity flow problem, which we optimize using the metaheuristics Adaptive Large Neighborhood Search (ALNS) and Greedy Randomized Adaptive Search Procedure (GRASP). Our scheduler supports scheduling of arbitrary communication-graphs on arbitrary topologies. We detail performance measurements and results for bi-torus and mesh topologies and show that we out perform a prior heuristic scheduler.

General Terms Static scheduling, Metaheuristics, Network-on-Chip, Real-time, Multi-commodity flow problem

Rasmus

1. Introduction

This report is part of the work carried out in the course 42137 OPTIMIZATION USING METAHEURISTICS. Each section heading is annotated with margin notes indicating authorship. All three authors has contributed equally to this project and report.

The goal of this project is to implement a scheduler for static inter-processor communication in a real-time Multi-Processor System-on-Chip (MPSoC), the scheduler should use metaheuristics for optimization of a greedy solution. In this type of Real-time system scheduling is done at compile-time to simplify Worst-Case Execution Time (WCET) analysis. In real-time systems, performance depends purely on the Worst-Case Execution Time (WCET) – therefore the analyzability of the system is very important to obtain good performance.

An MPSoC is built of *tiles* consisting of a processor, a router and links to neighboring tiles. A sample sketch of a tile can be seen in Figure 1. In a real-time MPSoC, each tile processor executes one task to keep the timing analysis simple and get a lower WCET bound.

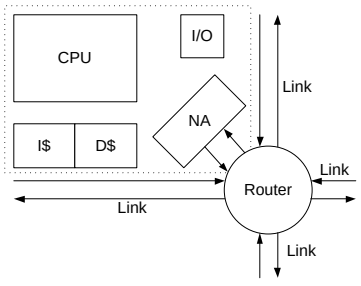
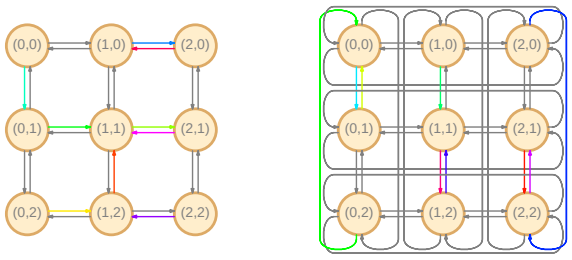


Figure 1: A tile contains a processor, a Network Adapter (NA), a local instruction and data cache and possibly some I/O.

An inter-processor communication channel is a point-to-point connection from one tile to another, this point-to-point connection can route packets along different paths. A path is the sequence of links, on which the packet travels to reach the end-point of the connection. Communication is statically scheduled such that communication on two channels can not interfere with each other – the communication channels are decoupled. Decoupling is required to make the NoC-interconnect analyzable with respect to WCET. The schedule is periodic, and thus for performance reasons (bandwidth and latency), we optimize for the shortest possible schedule period. Generating the optimal routing schedule is a hard problem to solve, for network topologies of a certain size generating an optimal schedule becomes impossible on the computer systems of today. The scheduling problem, has multiple sources and sinks making it a multi-commodity flow problem. The nature of the problem does not allow fractional flows on edges, thus the multi-commodity flow problem becomes NP-complete[3]. In Figure 2 we show a example of the first time slot of a 3×3 mesh schedule generated by our scheduler, the figures are auto generated by the scheduler.



(a) Mesh (b) Bi-torus

Figure 2: The first time slot of the schedule. The color indicates which communication channel is routed on the link.

Our goal for this project is to make a scheduler that makes as good schedules as possible within a reasonable amount of time. A reasonable amount of time should in this case be comparable to the design time of an embedded system, which might be in the range of days. The scheduler is meant to be usable for research purposes, thus we have constructed it very generic. Our scheduler can schedule arbitrary communication graphs on arbitrary interconnect topologies, the topologies are limited to routers with out degree 5, where one of the output ports always are connected to a CPU.

It is trivial to construct a greedy solution for the scheduling problem. To improve schedules compared to this greedy solution, we can apply metaheuristics.

Our scheduling problem is large and combinatorial in nature with both dependencies and constraints. A natural constraint is that any directed link can only transfer a single packet at any time-slot. Another constraint is that packets must arrive in the same order as they were sent. To avoid excessive feasibility-checking of the schedule due this in-order requirement of packets, we make the simplifying decision that all paths should be a shortest path. Due to the regularity of the graph, this shortest path routing is simple. Always taking the shortest path will automatically guarantee causality of received packets.

Our scheduling problem is NP-complete, thus we need metaheuristics to improve schedule periods from greedy algorithms. For most relevant network sizes it is practically impossible to search though the whole solution space.

Rasmus

2. Metaheuristics

To construct at good solution we expect that a greedy algorithm will give fairly good results, to improve the greedy solution metaheuristics can be used. A given greedy solution might not be a good target for optimization, due to the high density of the solution. A less dense solution might lead to a better solution.

Our intuition on how a dense solution can be improved, tells us that it has to be spread out before it can be compressed further.

Mimicking nature we think of the free links in the network as air bubbles. We then need to make room for the air bubbles to surface, this can be done by decompressing the schedule. To conclude, we should not only accept better solutions than our current.

Rasmus

2.1 Proposed metaheuristics

We expect the following two metaheuristics to yield good results:

- Greedy Randomized Adaptive Search Procedure (GRASP): Make greedy randomized initial solution. Make local search for better solution. Start over with new initial solution.
- Adaptive Large Neighborhood Search (ALNS): Choose initial solution. Destroy and rebuild part of current solution.

Jaspur

2.2 Initial solution

Four different approaches to create a feasible initial solution to optimize are:

Basic (BASIC) This initial solution creates a feasible schedule by randomly picking channels to be scheduled onto the NoC, and then schedules them on timeslots one after another. This solution only has one channel scheduled at each timeslot, and the advantage using this approach, is that the schedule is not dense and thus the applied metaheuristic will perform all the work towards an optimal solution.

Deterministic (GRREDY) The created deterministic approach is a greedy algorithm that tries to schedule channels onto the NoC at the earliest possible timeslot. The channels are scheduled in an order, where the longest channels are scheduled first. Length of channels is defined, by the number of links it has to take to reach its destination. The order, in which availability of links out of a router is checked, is also set. Due to the given attributes, this inital solution always gives the same result when given the same input. Therefore, using this initial solution, the metaheuristics are forced to try and optimize from the same starting point every time.

Randomized-routing (rGREEDY) This initial solution uses the same greedy approach as that of the deterministic, in that it tries to schedule the longest channels first, and the channels are scheduled at the earliest possible timeslot. The difference from the deterministic greedy algorithm is in how each channel is routed. Where the deterministic algorithm will always route the channels in the same way, the priority of possible available out links at each router is random, and thus if some routes can be routed better by using a different shortest path, this initial solution may find that route. Also, this gives an initial solution to the metaheuristic that is not always the same.

Randomized-routing and -order (RANDOM) Here, again, the channels are scheduled as early as possible, but now the order is random in which the channels are scheduled. This is another initial solution, where the starting point for the metaheuristic is a somewhat dense schedule, but since the routing ordering is random, using this initial solution will explore solutions different than those when deterministic or only randomized routing is used.

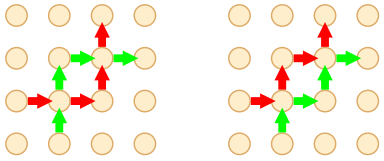
Mix between random and greedy (CROSS) Specially for GRASP, all the initial solutions we construct are a mix between RANDOM and rGREEDY. This is done by constructing a sorted list of the channels from the longest to the shortest, as done in GREEDY and rGREEDY. A float β between 0 and 1 is then passed as parameter. β denotes the percentage of how many paths should be swapped in the list of sorted-by-length channels. Hence β is a measure of how randomly the order in which channels should be routed; if $\beta = 1$ we have essentially RANDOM, if $\beta = 0$ we have rGREEDY.

The main motivation in having different ways of obtaining initial solution is to explore different neighborhoods with having different starting points.

2.3 Neighborhood and operators

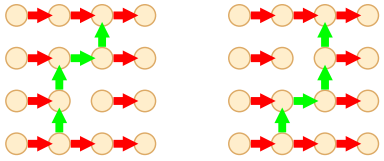
Mark

Our scheduling problem is to map a collection of channels to paths throughout time, without communication conflicts on any links at any time. A schedule with such conflicts is not feasible. The neighborhood is the search space around a current feasible solution when taking a small step away from it, i.e. incremental perturbations. Small perturbations of a current solution could be swapping pairs of links (Figure 3) or moving bends (Figure 4).



(a) From (b) To

Figure 3: Swapping pairs of links. This does not destroy feasibility, but has no advantage if both channels are present.



(a) From (b) To

Figure 4: Moving bends. This may not be feasible.

2.3.1 Primitives

Mark

Swapping pairs of links is always possible, but moving bends can lead to an infeasible schedule. Note that very little can be gained

by swapping pairs of links – only if one of the channels of Figure 3 is absent may there be a slight advantage. As the bend-moving example hints at, there is very limited freedom of movement in dense schedules. Hence we expect most local perturbations lead to infeasible schedules. Even if such small local perturbations could make room for other channels to be scheduled earlier, these small changes alone are unable to change period of the schedule. So instead of taking small steps, we would rather take large jumps. Instead of swapping of pairs or moving-bends as our primitive functions, we consider removing entire already-scheduled channels (`rip-up`) and rebuilding (`route`) them. Given a channel, `rip-up` deletes the entire already-scheduled channel from the schedule. Given an unscheduled channel, `route` will greedily (making randomized shortest-path routing decisions) try to schedule the channel as early as possible. `route` does this via a simple depth-first search. Of the paths which has just been ripped-up, the longest paths are routed first.

This approach of rerouting entire channels is obviously a superset of the small local perturbations. Thus the rerouting-approach greatly increases our neighborhood and increases our chances of finding a new and feasible path. Thus our neighborhood of a current schedule is the space formed by how many and which channels we can reroute. The "intelligent" selection of which channels to reroute is the job of our operators.

Mark

2.3.2 Operators

Which element from our neighborhood to jump to next, is determined by our operators. Several operators have been implemented, as this is required to make LNS adaptive. Since our primitive functions are always `rip-up` and `route`, the operators are reduced to simply selecting which channels should be rerouted.

Dominating paths Recall that our main objective is minimize the schedule period. It is therefore natural to attack the paths which finish last, since they prolong the schedule. We call these paths for the *dominating paths*. However simply rerouting only the dominating paths, is likely to not improve the period: All paths are rerouted in a greedy fashion, starting as early as possible. We therefore consider more than just the dominating paths, we would also like to select the paths that prevents the dominating paths from starting earlier. An approximation of those paths preventing a dominating path P from starting earlier, is simply the paths below P over all time. I.e. we select all the paths that goes through a link before P goes through it. Even through this is a gross approximation, this operator has shown to be the most popular in ALNS.

Dominating rectangle Rather than just selecting the channels below P in time, we select all paths in the shortest-path rectangle containing all of P. This is written as a BFS search, only taking links which are on the shortest-path.

Late paths This is exactly like dominating paths, but also considering the paths finishing next-to-last, i.e. the paths finishing at time $p-1$ and $p-2$.

Random Simply selects a random set of paths. Also the number of selected paths is random: At least 2 paths are always selected, but up to 10% of all existing paths may be selected.

Mark

2.3.3 Choosing operators adaptively

For both ALNS and GRASP, the above operators are used to mutate our current schedule to obtain the next. Since it is not clear which operator is the best, we chose which operator to execute adaptively – this is done according to a simple scheme: We calculate the ratio r between the period of the previous schedule and the new current schedule; $r = prev/curr$. Thus multiplying the probability of the selected operator by r will automatically punish or reward it. If $r > 1$, the current solution is shorter than the previous, so we reward by increasing the probability of the same operator being selected in the next iteration. Experimentation showed that dominating-paths was much better than random and probability of random-selection dropped quickly. To combat this quick convergence we actually multiply by $\sqrt{r}$ rather than r.

2.4 Greedy Randomized Adaptive Search Procedure

Rasmus

With the nature of our problem, where there is not a clear local neighborhood, diversifying the search for solutions may be an advantage. Also, the problem is very large, and these properties favor the GRASP metaheuristic.

The GRASP metaheuristic consists of the steps, 1) greedy randomized adaptive construction of a feasible solution, followed by 2) a local search on this solution. These steps are iterated ad infinitum until time runs out. The local search might not improve the solution, hence we also check if the greedy initial solution is the best seen so far.

Our implementation of the GRASP metaheuristic is not completely in accordance with the published version: Recall that our operators destroy and repair a part of the solution and are chosen adaptively – which is very similar to the ALNS scheme. Our GRASP can thus be seen as a layer on top of ALNS. Also, the iterative improvement in the local search performs only a single iteration in our case. Finally, instead of building the initial solution randomized and adaptively, the initial solution is built using CROSS as described in Section 2.2. This allows the user to tune β.

2.5 Adaptive Large Neighborhood Search

Mark

Contrary to GRASP, ALNS continually works on the same solution. ALNS repeatedly destroys a part of a solution and rebuilds it again into a feasible solution. Since we always rebuild with route, what we punish and reward are the selection operators described above. The operators fit into this scheme perfectly, so our implementation of ALNS is simple. Our ALNS implementation is as follows:

1. An initial solution is constructed using one of the specified algorithms described in Section 2.2. This is specified by the user.
2. One of the selection operators is chosen stochastically as described in Section 2.3.3.
3. The selection operator is executed, which returns a set of paths. This set of paths is ripped-up and re-routed again. As mentioned before, re-routing paths is done greedily: We first sort the ripped-up paths by length and route them according to this ordering. Any path is routed in a first-fit fashion, meaning we schedule it as early as possible. Also note that tie-breaking for shortest-path routing decisions is done stochastically.
4. The new schedule length is found, and if it the shortest seen until now, we copy it and remember it as our best solution thus far.
5. The selection operator we chose in step 2 is punished or rewarded as described in Section 2.3.3.

3. Implementation

Rasmus

Our scheduler is written in C++11, using BOOST 1.49[1] and pugixml 1.0[2] as 3rd party libraries. The source code of our schedule is provided as open source software. The source is available at `https://github.com/rbscloud/SNTs`. The source code builds

[1] `http://www.boost.org/doc/libs/1_49_0/`

[2] `http://pugixml.googlecode.com/files/pugixml-1.0.zip`

successfully on Linux and Cygwin with GCC 4.5.2 or later with `--std=c++0x` flag enabled.

The rest of this section will remain language agnostic.

Mark

3.1 Data-structure organization and delta-evaluation

In our problem domain, there are basically two ways to represent the schedule; time-centric or network-centric:

Time-centric We could consider grouping all routing-decisions made in the same timeslot, into a frame. The entire schedule would then consist of a layering of such frames, numbered from 0 through $period - 1$. We would have to be able to index into a frame given a router and its link, asking whether any channel in this frame has been scheduled on the link. However the interconnection network can have holes, and could be sparse.

Network-centric In reality, links are resources for which communication must contend. Thus is its more intuitive to have each link and router represented once only – rather than multiple times in each frame. This means we localize the schedule, storing a local schedule on each link. In this representation, this means we do not have to go up one timeslot and index again to check if something has been scheduled – this information is local. Splitting the schedule up into such localized schedules, also allows us to make fast successor queries. A drawback is that determining global schedule length now takes $\mathcal{O}(n)$ time where n is the number of routers, rather than $\mathcal{O}(1)$ time as in the time-centric representation. However using delta-evaluation this is reducible to average-case $\mathcal{O}(1)$ time.

The goal of delta-evaluation is to reuse as much of what is known about the previous solution, so the evaluation function becomes quick. In our case, we could store a dynamically re-sizable array which for each index (representing a timeslot), stores how many links is scheduled in that timeslot. Whenever we mutate our schedule, the array is updated correspondingly. Traversing backwards from the end of the array until hitting a non-zero, reveals the schedule length. Since the period of neighboring schedules is quite close to each other, traversing the array will take $\mathcal{O}(1)$ time in the average case, but $\mathcal{O}(p)$ time in the worst-case. Finally, resizing the array dynamically takes amortized constant time.

In reality, the overall performance depends on channel-routing and very little on our evaluation function, which is very simple and quick enough. Hence we have chosen the more Object Orientation-friendly network-centric representation. If further performance improvements are sought, our `route` primitive can be easily parallelized.

Mark

3.2 Calculating shortest paths

One of our simplifying decisions is to only route channels along shortest paths, as this will automatically guarantee in-order packet arrival. For this, we need to compute local routing decisions for the all-pair shortest paths. Since our network graph is very regular as each router only has max out-degree 4[3] and all links have unit cost, we can easily find the shortest path via a breadth-first search: We simply start at the destination b and do a BFS – as we encounter another router a, we now know a shortest path from a to b. That the path is a shortest path is guaranteed by the nature of BFS. Encountering a, we simply store which port the BFS-wave reached a from, and associate b with that port in a local (perfect) hashmap, $next$. So $a.next[b]$ returns a set of output ports of a, which lie on one of the shortest paths towards b. Computing all these shortest-paths can be done in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ space.

4. Parameter tuning

Jaspur

Parameter tuning is vital, in order to improve the level of optimization by the metaheuristics. In our case the main parameters to consider are:

Running time The time that is given to the metaheuristic to try and optimize the problem

Starting parameter of choose table The initial priorities of the operators applied by the metaheuristics

GRASP randomness β The parameter determining the randomness of the initial solution when running GRASP

Initial solution The chosen initial solution when running ALNS

As mentioned in the introduction the run time of our scheduler might be in the range of days, so proper parameter tuning, has not been possible in the duration of this project. We have restricted the run time of our scheduler to 2 hours, to get a first impression of the performance of the metaheuristics and their parameters. These reduced runs will be referred to as the test runs. During testing, we chose to focus on the mesh and bi-torus topologies with all-to-all communication. We focused on network instances of height and width 3 to 10 as well as 15 by 15.

4.1 Starting parameter of choose table

Jaspur

The starting parameters of the choose table denote the starting priorities of the available operators for the metaheuristic. Since we do not know which of the operators (dominating-{path,rectangle} and late-path) is best, implies that they should be weighted equally at the start of optimization. The adaptive function will favor those that give good results and punish the others. We have not tested other scenarios than equal weighting of these starting parameter in the choose table. We have assigned different values to the random operator in ALNS. The reason is that it is not as likely that the random operator is going to improve the schedule. Therefore, in order to force the ALNS metaheuristic to explore different neighborhoods, we ended up setting the starting parameter for the random operator to 33%, while the others are set to 22%, thus slightly favoring random initially. Preferably, with more time, we would vary the starting weights of the operators an observe the performance of the metaheuristics. For example, to achieve some kind of Simulated Annealing-like behavior we give the random-selection operator a higher probability initially. We could also elaborate our punish and reward system of operators, by also taking into account if the same schedule length is found.

4.2 GRASP randomness β

Jaspur

When running optimization using GRASP, we tested the algorithm by running for two hours with:

$$\beta \in \{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.5, 0.9, 1\}$$

Tables 1 and 2 show the best found schedule length and the standard deviation for each β. The standard deviation is calculated from samples of the iterations. To avoid excessive file I/O, the scheduler reports at most 1 data point each second. Thus for 2 hour runs we get at most 7200 data points. In the two tables, we have indicated the best solutions for the given network size in bold.

In table 1 we see that when running the GRASP metaheuristic on the bi-torus topology, a small amount of randomness is preferred. This is seen, since the best solutions for all the bi-torus sizes are found with $\beta \leq 0.05$. By the best solution we mean the β value that gets the smallest schedule length, and secondly has the smallest standard deviation.

[3] For routing we only care about the 4 non-local ports of routers.

Size \ β	0		0.01		0.02		0.05		0.1		0.5		1	
	min	σ	min	σ	min	σ	min	σ	min	σ	min	σ	min	σ
3×3	11	0.47	11	0.47	**10**	**0.55**	10	0.57	10	0.56	10	0.59	10	0.59
4×4	**19**	**0.57**	19	0.62	19	0.60	19	0.61	19	0.67	19	0.66	19	0.69
5×5	30	0.70	**30**	**0.69**	**30**	**0.69**	30	0.74	30	0.76	30	0.79	30	0.79
6×6	43	0.88	**43**	**0.86**	43	0.98	43	0.94	44	0.97	44	0.93	44	0.91
7×7	61	1.14	61	1.08	**61**	**0.95**	61	1.08	61	1.13	62	1.08	62	1.10
8×8	85	1.26	**85**	**1.15**	85	1.37	85	1.33	85	1.42	85	1.44	86	1.35
9×9	113	1.80	113	1.60	113	1.60	**113**	**1.44**	114	1.58	115	1.53	116	1.53
10×10	151	2.08	151	2.00	152	2.13	**151**	**1.90**	152	1.87	153	1.99	155	1.95
15×15	**474**	**6.81**	477	2.46	479	2.46	481	3.30	485	3.59	501	4.00	506	3.42

Table 1: Bi-torus: Best found solution and standard deviation running GRASP, for the different β values and bi-torus sizes. The columns for 0.2, 0.3 and 0.9 are omitted to lack of information of interest.

Size \ β	0.01		0.02		0.1		0.2		0.3		0.5		1	
	min	σ	min	σ	min	σ	min	σ	min	σ	min	σ	min	σ
3×3	**11**	**0.57**	11	0.59	11	0.60	11	0.62	11	0.62	11	0.64	11	0.64
4×4	22	0.76	**21**	**0.75**	21	0.78	21	0.84	21	0.81	21	0.79	21	0.77
5×5	39	0.98	38	1.01	38	1.06	37	1.13	37	1.14	37	1.09	**37**	**0.99**
6×6	63	1.24	63	1.33	62	1.60	61	1.27	61	1.44	**61**	**1.18**	62	1.12
7×7	97	1.49	96	1.77	95	1.92	**94**	**1.63**	94	2.18	95	1.78	95	1.93
8×8	141	2.47	140	2.21	139	2.39	**138**	**1.63**	139	2.44	139	2.12	139	2.41
9×9	198	3.15	198	3.01	196	2.70	**195**	**2.63**	195	3.11	196	3.15	198	2.83
10×10	270	3.84	269	3.48	**267**	**3.17**	267	3.39	267	3.31	268	3.31	269	3.53
15×15	903	3.76	903	7.58	**899**	**5.12**	899	7.18	902	5.98	905	8.28	913	9.28

Table 2: Mesh: Best found solution and standard deviation running GRASP, for the different β values and mesh sizes. The columns for 0, 0.05 and 0.9 are omitted to lack of information of interest.

For the mesh topology, table 2 shows, that the best results can be found across the table. We observe that $\beta = 0.2$ results in the best schedule lengths with a relatively good standard deviations. From this we gather, that in contrary to the bi-torus topology, the mesh topology gets the best schedule length when some randomness is added to the initial solutions.

It is an interesting observation that the β values for bi-torus where the scheduler performs best are lower than the β values for the mesh where the scheduler performs best. This could be because the bi-torus topology is more regular than the mesh topology.

Jaspur

4.3 Initial solution

ALNS starts with an initial solution and tries to improve it. How this initial solution should be created is not given intuitively. We show how different initial solutions makes the ALNS perform. When doing a test run of 2 hours, the schedule length for the different sizes of the bi-torus topology is seen in table 3.

Bi-torus	BASIC	GREEDY	rGREEDY	RANDOM
3×3	11	11	11	**10**
4×4	**19**	**19**	**19**	**19**
5×5	**30**	**30**	**30**	**30**
6×6	**45**	**45**	**45**	**45**
7×7	**63**	**63**	**63**	64
8×8	87	87	**86**	88
9×9	118	**113**	115	119
10×10	160	**153**	155	158
15×15	514	**471**	477	507

Table 3: Bi-torus: Shortest schedule period for bi-torus, when running ALNS one time on different initial solutions and sizes of networks for two hours.

In table 3, we see that up to 7×7, all the initial solutions perform almost equally. The length of the BASIC initial solution will be longer than the other initial solutions, but this might avoid that our search falls in a local minimum. The RANDOM solution is the only one to differ slightly from the others, in finding a better solution for size 3×3 and performing worse for 7×7. For size 8×8 the initial solution using randomized routing found the best schedule. For larger networks the deterministic GREEDY performs best. Since using GREEDY as initial solution performs best in all but two cases, this implies that generally it is best solution to start from for ALNS when optimizing the schedule for a bi-torus. The same observations are made when looking at the same numbers for the mesh topology.

5. Results

Rasmus

Our scheduler has been tested with custom topologies besides mesh and bi-torus, to verify functionality. For evaluating the performance of our metaheuristics we only run our scheduler on use-cases with all-to-all communication and on either a mesh or a bi-torus topology these are the only examples for which we have results to compare with. For all-to-all communication in a mesh or bi-torus topology we also have theoretical lower bound on the scheduling period.

Lower bound for the scheduling period are given in [1]. In [2] a heuristic scheduler is proposed, this scheduler has the constraint compared to our scheduler that the schedule in each router must be the same. This constraint is made to simplify hardware. Table 4 compares our results to the results of the heuristic scheduler in [2] and the lower bounds given in [1], the numbers in parenthesis in the bounds column are optimal schedule periods also given in [1]. These results have all run for two hours. It takes 504 CPU-hours (or 21 CPU-days) in total to run all our test-runs; on gramme this takes just 15 hours.

Size	Mesh					Bi-torus				
	Bounds[1]	[2]	GREEDY	ALNS	GRASP	Bounds[1]	[2]	GREEDY	ALNS	GRASP
3×3	8 (10)	28	13	**11**	**11**	8 (10)	11	12	**10**	**10**
4×4	16 (18)	59	24	**21**	**21**	15 (18)	20	21	**19**	**19**
5×5	25 (34)	112	41	39	**37**	24 (28)	**28**	32	30	30
6×6	54	–	66	65	**61**	35	–	45	45	**43**
7×7	66	–	98	97	**94**	48	–	64	63	**61**
8×8	128	481	144	144	**138**	64	88	87	86	**85**
9×9	135	–	201	201	**195**	90	–	**113**	**113**	**113**
10×10	250	974	271	271	**267**	125	158	154	153	**151**
15×15	600	3467	**886**	**886**	899	420	481	**471**	**471**	474

Table 4: Results compared to the heuristic results of [2]. Numbers in parenthesis are optimal schedule periods

In table 4 we see that our scheduler performs very well. Our scheduler is better than the results of [2] in all cases except 5×5 bi-torus. We are much better than [2] for all sizes of mesh networks. The reason for this improvement is the constraint, that all routers should have the same schedule. For the bi-torus our schedules are within approx. 30 % of the analytical bound from [1]. As seen in the bounds column in table 4 the optimal schedule period for a 3×3 bi-torus network is 10, as is the result of both our ALNS and GRASP schedules. Our scheduler is not fare from the optimal schedule periods for small systems.

Comparing our greedy solution to our metaheuristic solutions, we see that our metaheuristics can improve our greedy solution for most cases. For the large network sizes the improvement by the metaheuristics diminish. Two reasons for these diminishing improvements are the increasing link utilization seen in figure 5 and the decreasing iteration count seen in table 5.

The link utilization increases because the average communication channel length grows with the network size. The link utilization for the greedy bi-torus schedules, as a function of the network size, can be seen in figure 5. As the link utilization increases intuition tells us that compressing the greedy solution becomes increasingly difficult, thus the metaheuristics need to put in more effort to optimize the schedules.

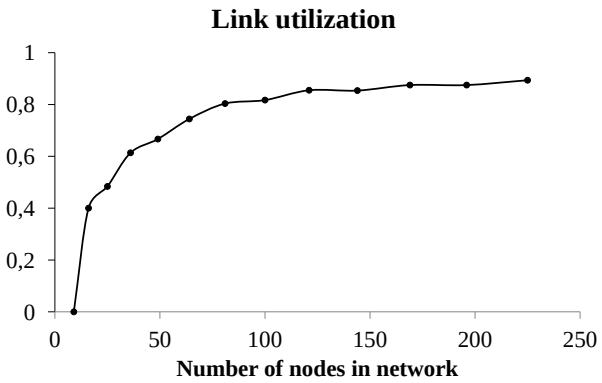


Figure 5: Link utilization of the greedy schedule on a bi-torus topology.

When we look at the iteration count in table 5 we see that for the large network sizes the number of iterations are very small, this means that the scheduler, for these sizes, should run for a longer period of time. Due to the low iteration count of the large network sizes, we believe that running the scheduler for a longer period of time can still improve the results to be better than the greedy solution. Although we do not expect to get the same fractional improvement as for the smaller network sizes, due to the increased link utilization.

Size	ALNS		GRASP	
	Bi-torus	Mesh	Bi-torus	Mesh
3×3	466.451.877	271.864.914	33.164.408	30.629.712
4×4	125.812.628	100.497.005	6.997.571	5.486.080
5×5	87.230.064	29.898.319	2.555.017	1.886.940
6×6	39.682.545	10.825.917	823.677	457.601
7×7	19.926.148	4.270.345	247.464	159.557
8×8	6.382.482	1.478.727	123.695	55.597
9×9	3.205.896	1.478.727	64.213	21.638
10×10	1.350.495	353.699	27.495	11.543
15×15	15.969	6.944	1.070	84

Table 5: Numbers of iterations, for a single run for two hours on gramme.ebar.dtu.dk. The initial solution used by ALNS is RANDOM and GRASP uses $\beta = 0.1$.

Combining the increasing link utilization and the decreasing iteration count, says that we should increase the run time for large network sizes considerably.

6. Conclusion

In this report we have shown how to apply two metaheuristics Adaptive Large Neighborhood Search and Greedy Randomize Adaptive Search Procedure to an NP-complete scheduling problem. We have shown that our scheduler gives results close to the optimal solution for small network sizes and for larger network sizes we are within ~30 % of the analytical lower bound for bi-torus. The results for the large network sizes can be expected to improve when the scheduler is allowed to run for several days. We have shown that our scheduler performs better a prior heuristic scheduler. For large mesh networks our scheduler is a factor of 3.9x better. Overall GASP has shown to be the best metaheuristic for this scheduling problem.

References

[1] Martin Schoeberl et. al. A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems To be published at *IEEE International Symposium om Networks-on-Chip 2012(NOCS '01)* *http://www.jopdesign.com/doc/s4noc.pdf*

[2] Florian Brandner et. al. Static Routing in Symmetric Real-Time Network-on-Chips Not yet accepted, flbr@imm.dtu.dk

[3] Karp, R. M., On the complexity of combinatorial problems Networks, vol. 5, No. 1, 1975, pp. 45-68.