Scheduling in a Real-time Network-on-Chip

Period minimization using metaheuristics

Mark Ruvald Pedersen Rasmus Bo Sørensen Jaspur Højgaard

Technical University of Denmark, Kgs. Lyngby

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

1. Introduction

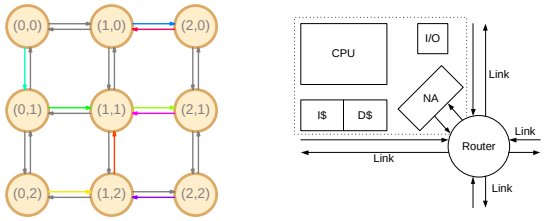
For the T-CREST[1] project, a real-time Network-on-Chip (NoC) is to be designed. We have implemented a scheduler for static inter-processor communication in a real-time Multi-Processor System-on-Chip (MPSoC). In this type of real-time system scheduling is done at compile-time to simplify Worst-Case Execution 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 1b. In a real-time MPSoC, each tile processor executes one task to keep the timing analysis simple and get a lower WCET bound.

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 1a 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.

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 Network Adapter.

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.



(a) A time slot of a schedule. Color indicates which communication channel is routed on the link.

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

Figure 1

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.

2.1 Neighborhood and operators

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 of links 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. The longest of the ripped-up 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

[1] http://www.t-crest.org/

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

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. The most successful operator is our dominating paths: We call the paths that finish last for the dominating paths, and includes all paths below them in time. It is natural to attack these paths since they prolong the schedule.

2.2 Greedy Randomized Adaptive Search Procedure

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.

2.3 Adaptive Large Neighborhood Search

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 the routes, we punish and reward the selection operators. The operators fit into this scheme perfectly, so our implementation is close to standard ALNS. Our adaptive scheme is based on multiplying the operator-probability with the improvement.

3. Results

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 1 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.

In table 1 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 1 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 and the decreasing iteration count, which spans more than 4 orders of magnitude.

4. 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.