Pre-Hand-In
42137 Optimization using Metaheuristics

Jaspur Højgaard - s072069
Rasmus Bo Sørensen - s072080
Mark Ruvald Pedersen - s072095

March 18, 2012

1 Problem outline

In this project we are to implement a meta heuristic for scheduling communication in a real-time Multi-Processor System-on-Chip (MPSoC). The MPSoCs we will be targeting for this project, are MPSoCs with a Network-on-Chip (NoC) as interconnect. This general-purpose real-time MPSoC is part of the T-Crest project[1] funded by the European Union.

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. A NoC 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.

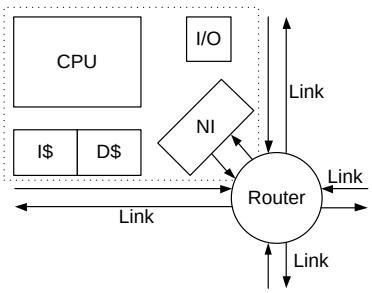
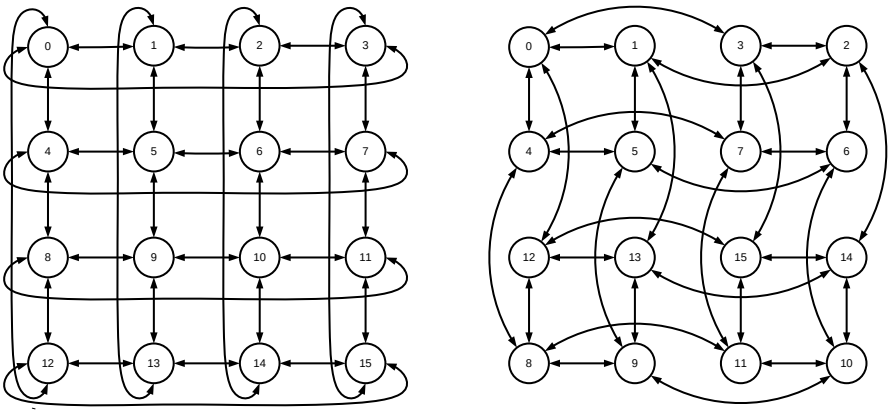


Figure 1: A tile contains a processor, a Network Interface (NI), a local instruction and data cache and possibly some I/O.

A communication channel is a point-to-point connection from one tile to another, this point-to-point connection can route packets along different paths.

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



(a) Nice graph representation.

(b) Minimized longest-link distance.

Figure 2: Two double-torus graphs, only differing in layout.

A path is the sequence of links, on which the packet travels to reach the end point of the connection. Two communication channels can not affect each other – they are decoupled. Decoupling is required to make the NoC-interconnect analyzable. To decouple communication channels, the traffic in the network can be scheduled statically at compile time. Generating this routing schedule is a hard problem to solve. The schedule is periodic, and thus for performance reasons (bandwidth and latency) it is important to have the schedule as short as possible. Given use-case, we therefore want to optimize towards the shortest schedule.

Routers can be connected in many different topologies, the most common structure is in a grid-like fashion. One possible NoC topology is a double-torus, an example of this layout is seen in Figure 2a. In this figure, some of the links are very long which does not scale well in hardware implementations. Figure 2b is an example of how the double-torus can be laid out, with a minimum average link length.

2 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 timeslot. Another 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.

2.1 Neighborhood and operators

The problem consists of a collection of paths to be scheduled onto the NoC. We consider the neighborhood of one feasible solution, to be all other feasible solutions where k paths are permuted. A permutation of a path could be patching a part of it (local changes, eg. swapping individual links), or rerouting it in a randomized greedy fashion.

Imagine a compact schedule where most links are already occupied, we expect small changes like swapping individual links to result in a lot of infeasible schedules. Instead we consider removing entire paths and rebuilding them: Patching an existing path versus rerouting it, the rerouting-approach will obviously be a superset, as the patching-approach makes only locally limited changes. The rerouting-approach greatly increases our neighborhood and increases our chances of finding a new and feasible path. It is also possible to reroute part of an existing path.

We can use the above approach to "make room" in the schedule such that the latest-finishing path may be moved to an earlier starting timeslot – thus reducing the overall schedule period.

2.2 Proposed metaheuristics

We expect the following two metaheuristics to yield good results:

- Large Neighborhood Search (LNS): Choose initial solution. Destroy and rebuild part of current solution. This is exactly what we described above.
- Greedy Randomized Adaptive Search Procedure (GRASP): Make greedy initial solution. Destroy and rebuild part of solution. Start over with new initial solution.

Finally, we can implement Simulated Annealing – since it is simple and can provide a frame of reference.