Technische Universität München
Chair of Scientific Computing
Dr. Ralf-Peter Mundani
Dipl.Inf. Atanas Atanasov

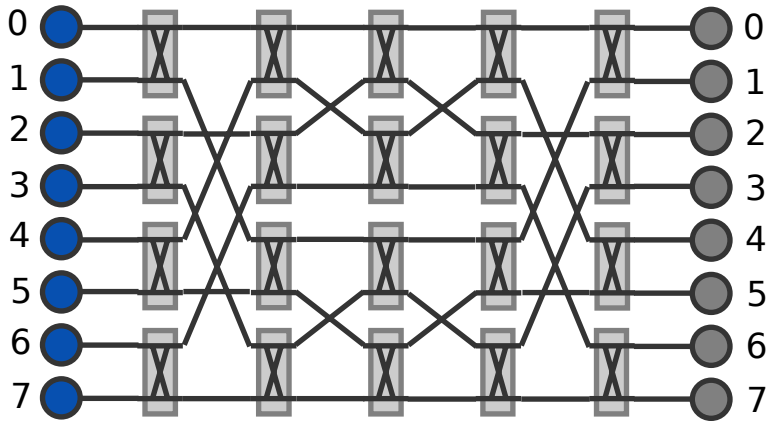
SS 2010

Parallel Programming and HPC Exercise Sheet 3: Topologies, Dependency Analysis and Synchronization

May 18, 2010

1 Contention-Free Routing: Beneš

One possibility of interconnecting the processors of a multi-processor system with a dynamic network topology is given by the so-called Beneš multistage network. The key property of this topology is, that for any permutation of inputs to outputs, there always exists a contention-free routing. Below is shown an example for a Beneš network with 8 inputs and 8 outputs.



a) Show the following non-blocking parallel communications for the network above:

$0 \rightarrow 3$	$1 \rightarrow 5$
$2 \rightarrow 4$	$3 \rightarrow 7$
$4 \rightarrow 6$	$5 \rightarrow 0$
$6 \rightarrow 1$	$7 \rightarrow 2$

b) Explain why the Beneš network is non-blocking.

2 Rearrangeable / Strict-Sense Non-Blocking: Clos

To reduce the total amount of necessary switching elements in case of crossbars for interconnecting K inputs with K outputs, Clos networks consist of three stages (ingress, middle, and egress) of much smaller crossbars, i.e. with less than K^2 switching elements. Hence, the ingress stage contains R crossbars with $N \times M$ inputs/outputs, the middle stage contains M crossbars with $R \times R$ inputs/outputs, and the egress stage contains R crossbars with $M \times N$ inputs/outputs. Depending on the relative values of M and N, rearrangeable non-blocking ($M \geq N$) and strict-sense non-blocking ($M \geq 2N - 1$) can be distinguished.

a) Draw a strict-sense non-blocking Clos network with properties $R = N = 2$. Show the following parallel communications:

$$0 \rightarrow 3 \qquad 1 \rightarrow 1$$
$$2 \rightarrow 0 \qquad 3 \rightarrow 2$$

b) Draw a rearrangeable non-blocking Clos network with properties $R = N = 3$. Show one situation in which rearrangement is necessary.

3 Dependency Analysis

For the successful parallel execution of processes, a dependency analysis is necessary. Here, Bernstein's conditions, for instance, help to identify instruction-level parallelism as to be observed in loops. It is required, that an iteration of a loop is independent from all other iterations, i.e. there do not exist read-write dependencies. Examine the following loop and decide whether instruction-level parallelism can occur or not.

```
for(i=1; i≤n; ++i){
      a[i] = 2 * a[i+m] + b[i];
}
```

For which values of $m \in \{-1, 0, 1, 2\}$ can the iterations be executed in parallel?

4 Baboon crossing problem

There is a deep canyon somewhere in Kruger National Park, South Africa, and a single rope that spans the canyon. Baboons can cross the canyon by swinging hand-over-hand on the rope, but if two baboons going in opposite directions meet in the middle, they will fight and drop to their deaths. Furthermore, the rope is only strong enough to hold 5 baboons. If there are more baboons on the rope at the same time, it will break. Assuming that we can teach the baboons to use semaphores, we would like to design a synchronization scheme with the following properties:

- Several baboons can cross at the same time, provided that they are all going in the same direction.
- If eastward moving and westward moving baboons ever get onto the rope at the same time, a deadlock will result (the baboons will get stuck in the middle) because it is impossible for one baboon to climb over another one while suspended over the canyon.
- If a baboon wants to cross the canyon, he must check to see that no other baboon is currently crossing in the opposite direction.

1. Simulate until 60 baboons have crossed each way, then stop.
2. Baboons arrive every 1, 2, 3 or 4 seconds, randomly coming from either the east or the west with equal probability.
3. Traversal of the canyon takes 3, 4, 5 or 6 seconds with equal probability.
4. All baboons currently on the rope crossing the canyon must be moving in the same direction.
5. Your solution should avoid starvation (i.e. making a baboon wait indefinitely). so, for example, when a baboon that wants to cross from the east arrives at the rope and finds baboons crossing from the west, he waits until the rope in empty, but no more baboons arriving from the west are allowed to start until at least one baboon has crossed the other way.

Implement your a multi-threaded application in Java, which solves the problem!