Parallel computing **8.2.2018**

Exercise 5

These exercises exceptionally at Joensuu on Thursday 8.2. 10-12 TB248, and at Kuopio on Monday 12.2. 14-16 F213.

Submit each solution separately to Moodle by 8.2. 09:00 (1 hour before exercises). Take skeletons from course www-page.

21. Parallelize the prefix-sum -algorithm using OpenMP. Take the sequential implementation from www-page and use the same technique that we showed in the blocking prefix sum algorithm and radix sort. I.e., do local prefix sums, prefix sum the sums (sequentially), and adjust the final result. Now the simple "`parallel for`" -construct is not enough, but you need to use OpenMP threads and make a function that uses those.

22. Refine the PRAM algorithm for parallel "binary search" (idea described on lectures). The algorithm returns the position (rank) of an element x in a sorted array A. Do not use CW variants. The algorithm is actually a $(P+1)$ -ary search as it divides the input with P division points on every iteration. Pay attention to the indexes and make the indexing work with any N or P.

23. Parallelize the [Sellers] dynamic programming algorithm for approximate string matching. First design a PRAM parallelization (O(textlength) time should be straightforward). Notice, however, the dependencies between cells.

24. Parallelize the C version of the dynamic programming approximate string matching algorithm using OpenMP. Now, as we cannot use strict synchrony, the parallelization is a bit more difficult. You need to find a way to make computation a bit more asynchronous. Remember that we have only few processors. Achieving speedup might be difficult, but keep the algorithm function correct. The sequential version is available at www-page.

Now we'll forget shared memory and start using message passing between processes/processors. All processes execute the same code. Use PID (process-id) to distinguish the processes.

25. Convert the prefix-sum algorithm (handouts p. 38, slides p. 126) to a message passing algorithm. In the beginning, each process has a single value. In the end, the each process has the corresponding value of the prefix sum. Express your algorithm so that all processes can execute the same algorithm, i.e., use PID (process-id) in giving roles to processes. Use message passing primitives `send(in destination, in data)` and `receive(in source, out data)`, where `destination` and `source` are process-id's and `data` is the value to be send / variable to which message is received.