Assignment No. 6

Parallel Computing, DM8XX (Fall 2008)

Department of Mathematics and Computer Science
University of Southern Denmark
Daniel Merkle

Due on: Thursday 18. December, 12:00 p.m. (Department secretaries office (Lone Seidler Petterson) or my office).

Note that you can achieve a **maximum of 30 points** for this assignment. (This was updates on Dec. 15th).

Exercise 1

`Floyd again` (40 points)

Compute the parallel run time, speedup, and efficiency of Floyd's all-pairs shortest paths algorithm using 2-D block mapping on a p-process mesh with store-and-forward routing and a p-process hypercube and a p-process mesh with cut-through routing.

Exercise 2

`0/1 Knapsack` (20 points)

Consider the parallel algorithm for solving the 0/1 knapsack problem in Section 12.2.2 of the course book. Derive the speedup and efficiency for this algorithm (as a function of n, c, p, t_c, t_s, and t_w. Show that the efficiency of this algorithm cannot be increased beyond a certain value by increasing the problem size for a fixed number of processing elements. What is the upper bound on efficiency for this formulation as a function of t_w and t_c?

Exercise 3

`0/1 Knapsack` (60 points)

Implement a parallel version for solving the 0/1 Knapsack problem with the dynamic programming approach as introduced in the lecture. Measure, as usual, the efficiency based on 10 test runs. Use weights and profits as random integer numbers between 1 and 50. The number of objects $n \geq 1$ and the maximal number of nodes $p \geq 1$ to be used should be given as input parameter. The program should iterate over all possible combination of n and p and print out the corresponding efficiency.

Exercise 4

`Pancake Sorting` (60 points)

Note: This exercise will probably require several hours of programming. If your time is limited, please chose other exercises.

Pancake sorting is a variation of the sorting problem in which the only allowed operation is to reverse the elements of some prefix of the sequence (see for example `http://en.wikipedia.org/wiki/Pancake_sorting`). For example the permutation $(2, 3, 4, 1)$ can be sorted by two prefix reversals. After the prefix reversal of $2, 3, 4$ we achieve $(4, 3, 2, 1)$, and then we invert the complete permutation and we achieve $(1, 2, 3, 4)$, which, by definition is the sorted permutation.

Implement a parallel version that computes the minimal number of prefix reversals needed to sort a given permutation by a brute force approach via a depth first search. Feel free how to implement and to parallelize your program. The number of nodes to be used should be given as a parameter, and the input permutation should be read from a file.