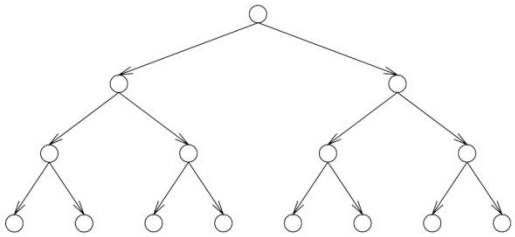
Prof. Dr. C. Stamm christoph.stamm@fhnw.ch Tel.: 056 202 78 32

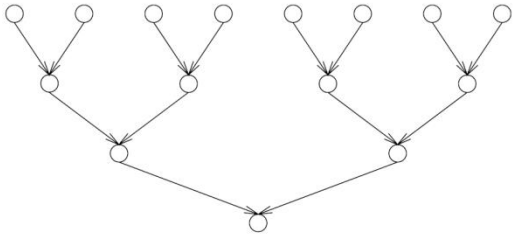
# Exercise 4

## 1 Task Graphs

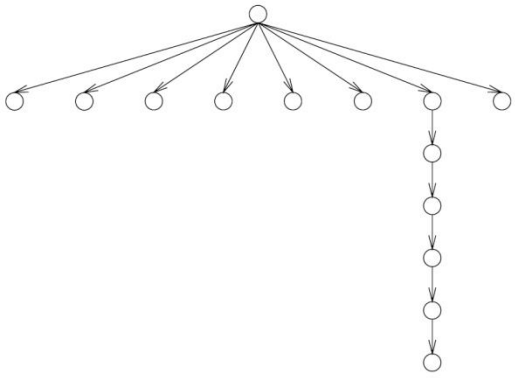
Given are the following four tasks graphs. Assume each node to be of unit weight.



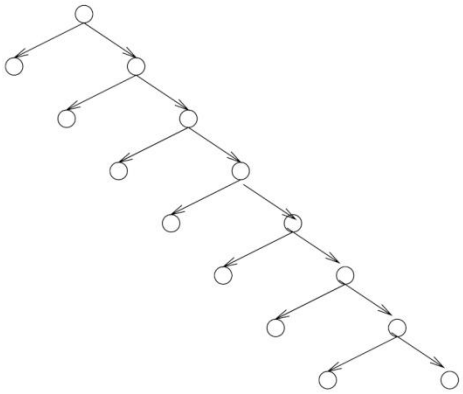
(a)



(b)



(c)



(d)

For the four task graphs, determine the following:

a) Maximum degree of concurrency.
b) Critical path length.
c) Maximum achievable speedup over one processor if an arbitrarily large number of processors is available.
d) The minimum number of processors needed to obtain the maximum possible speedup.
e) The maximum achievable speedup if the number of processors is limited to (i) 2, (ii) 4, and (iii) 8.

## 2 MPI: Odd-Even Transposition Sort

Install MPI[1] on your computer and make sure you can run the given MPI program "OddEvenSort.cpp" on several processes in parallel on a single machine. If everything works well, then study and modify the program in the following ways:

a) Study the (parallel) algorithm odd-even transposition sort described in Wikipedia[2] or on slides 10.6–10.7. Compare the provided implementation with the description on the slides. What is different?

[1] I recommend the MPI implementation of MPICH: http://www.mpich.org/downloads/. You will need both mpich-x.y and hydra-x.y. Windows user should download msmpisdk.msi and msmpisetup.exe from https://www.microsoft.com/en-us/download/details.aspx?id=54607.
[2] Wikipedia Odd-even sort: https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort

b) What is the parallel time complexity $T_P$ of the implemented algorithm? Derive expressions for the parallel run time, efficiency, and isoefficiency function. What is the maximum number of processes that this parallel formulation can use to be cost-optimal?
c) Measure the wall-clock times of all processes with MPI_Wtime() and inform process 0 about the largest wall-clock time. Process 0 should report the largest wall-clock time on console. Ideally, all of the processes would start execution of the local sorting at the same time, and then you could report the time that elapsed when the last process finished. Unfortunately, you can't get exactly this time, because you can't ensure that all the processes start at the same time instant. However, you can come reasonably close if you use a barrier synchronization immediately before each process starts time measuring.
d) In the given program each process initializes its own local elements with random values. Modify the program so that only process 0 allocates an array of size *n* and initializes all elements with a random value. Then the array is equally partitioned and mapped on all processes. Each process locally sorts its partition and starts parallel odd-even sorting. At the end of the sorting process each process sends its part to process 0 and process 0 checks the order of all *n* elements. Compare the wall-clock times of an efficient serial sorting algorithm with your implementation of parallel odd-even transposition sort and compute the speedup and the Karp-Flatt metric for different number of processes.

# 3 MPI: Numerical Integration

In numerical analysis, numerical integration constitutes a broad family of algorithms for calculating the numerical value of a definite integral. In this exercise you should approximate the definite integral $\int_0^1 \frac{1}{(1+x^2)} dx$ by the following two algorithms[3]. Let the user enter the number of subintervals *n* and send this number in an efficient way to all processes. At the end, process 0 should show the result of the numerical integration. What is the expected result?

a) *Rectangle Rule:* Divide the interval [0,1] in *n* subintervals of width 1/*n* and equally partition the resulting *n* sub-intervals to processes. Each process computes then the sum of the piecewise rectangular areas of the assigned subintervals and process 0 finally collects and sums up these partial sums. What partitioning scheme is useful to achieve a good load balance?
b) *Trapezoidal Rule:* Divide the interval [0,1] in *n* subintervals of width 1/*n* and use block partitioning. Each process computes for its partition the partial sum with the trapezoidal rule and process 0 finally collects and sums up these partial sums.

[3] Numerical integration: https://en.wikipedia.org/wiki/Numerical_integration