High Performance Computing: Sheet 6

Nils Döring Michael Mardaus Julian F. Rost
Sebastian Müller

10. Dezember 2013

Question 1

a)

For each combination of tours the algorithm has to check if its shorter than the existing ones. The number of tours lies in $O(n!)$ and therefore grows exceedingly fast. Each tour has to be stored before distribution which causes a high load in memory. Before `City_count(tour) == n` for the first time the queue is huge already. All complete tours are found on leaf level of the search tree.

b)

c)

Question 2

a)

If we half the stack only by the number of tours, we do not look for the time of execution needed for each tour. Due to the fact that every tour has different depth and therefore takes a different time to end, we don't balance the workload equaly. Also the stack of (incomplete) tours in process A after popping the $k/2$ top tours will very likely build exactly the same searchpath as it already had before popping.

b)

This strategy works better, but does not help, if there is just one job taking much time to execute, while the others are fast finished. Therefore one process, let it be process A, will take a lot of time, while the other is finished much earlier.

c)

For the execution time alone this strategy is good. The process-load is averagely distributed. But in order to calculate the average costs for each edge, a lot of computing time is required, because each edge has to be walked through. Therefore this strategy has a high build-up time, which was unlikely to be canceled out by the lesser time the DFSes would take.