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

Exercise 6

1 Shellsort

In Exercise 4 you have studied and modified an MPI based parallel version of odd-even transposition sort. Now you should implement the parallel version of Shellsort described on Slide 10.8, which uses in its second phase odd-even transposition sort. Make use of MPI and change the data type of the array elements from `int` to `float`, that you can compare the performance with your Quicksort implementation. Measure the performance of your implementation for varying array sizes $n = 2^{3k}$, $k \in [5, 9]$.

2 Quicksort

Given are the files “main.cpp” and “quicksort.cpp”. “main.cpp” contains a test environment and “quicksort.cpp” contains an already implemented sequential version of Quicksort (e.g. Slide 10.25).

a) Implement an efficient parallel version of Quicksort for shared memory systems. Let the number of processing elements p be an additional user defined parameter.
b) Compare the performance of your parallel Quicksort implementation with the performance of the serial and parallel standard sorting algorithms, e.g. `std::sort(…)`, `parallel_sort(…)`, and with your MPI implementation of Shellsort.

   Plot the performance for varying array sizes $n = 2^{3k}$, $k \in [5, 9]$, analyze and discuss the plotted curves.

3 Parallel Graph Search

In the last week of the semester we will try to connect our laptops to ad-hoc clusters, one for Windows and one for Linux based hosts. These clusters should give us a chance to test our MPI algorithms on a larger number of processes. As an exercise object, we will use parallel graph search, since this is very well suited to generate different computing loads.

In order to be able to work on task in the last week, you need the following preparations (a-c), which you should have completed before the lectures:

a) The given source code includes a widely usable parallel graph search framework for distributed systems and one application of the framework, the 15-puzzle (“Puzzle.cpp”). Build and run the application on your system, study the source code carefully, and answer the following questions:
   1) What type of graph is used in parallel graph search?
   2) Which search strategy is used in the framework?
   3) How is the open list implemented (what data structure is used)?
   4) What data is exchanged between searcher and master and what MPI data type is used?
   5) How is the solution path formed after a solution has been found?
b) Running MPI on a cluster requires on all hosts:
   1) identical user accounts (use “MPI” as user name);
   2) compatible MPI installations (MPICH, Microsoft MPI);
   3) your executable in the same local directory (Windows: %USERPROFILE%\Documents);
   4) a running smpd daemon.
c) If you have at least two hosts in a network, then you could already try to run the same MPI program on these connected hosts. Following steps are required:
   1) login as MPI user on all hosts and start smpd with option -d;
   2) choose one of the hosts as the master, build the newest version of the executable, and copy it to all hosts to the same local directory;
   3) on your master edit a host file (e.g. “hosts.txt”) that lists on each line a host name and the number of MPI processes separated by a blank;
   4) on your master save the following (adapted) command in a command script:
      ```
      mpiexec -machinefile hosts.txt -wdir %USERPROFILE%\Documents -lines Exercise6_MPI.exe
      ```
   5) run the command script on your master.
d) Exercise: Implement an admissible heuristic for the given 15-puzzle implementation.