Parallel Sorting Excercise

Prof. Peter Sanders, Vitaly Osipov, Yaroslav Akhremtsev

November 5, 2014

1 Problem Statement

Sorting is a basic operation involved in many algorithms as a subroutine. Therefore it is important to provide a fast sorting primitive that exploits parallelism available in current architectures. In this exercise we develop different sorting algorithms using different technologies for parallel programming, such as OpenMP, C++11 STL Threads library, Intel TBB, Open MPI*, Cuda*

2 Plan of Work

2.1 Preliminaries

- Split in groups of size at most 2.
- Get to know the GIT version system and register at `https://education.github.com/pack/join`. As a student you will get a possibility to create free private repositories on the github. All the source code should be submitted to github. Each member of the group has to submit under her/his own account.
- Please comment the code, so that we can see what the code does in the source files.
- Use your private laptops to debug the code, try to keep only the correct code in github.
- As soon as you push your code into the github it will be automatically uploaded into our continuous integration system, compiled, and undergo some correctness tests and benchmarks. The exact input/output formats will be formalized later. You will be able to see the console output of your program on our servers.

2.2 Naive Quicksort

As a warm-up we propose to implement a naive quicksort algorithm using a technology of your choice.

- Get to know how the algorithm works.
- Implement serial quicksort and use it as correctness test for your parallel implementation.
- Do not parallelize partitioning, only the recursive calls should be parallelized.
- Choose your favorite parallel programming technology and implement a parallel version of the algorithm.

2.3 Your Favorite Sort

- Choose your favorite sorting algorithm from: "non-naive" quicksort, inplace quicksort, merge sort, sample sort, radix sort. And send your preference list including all algorithms to osipov@kit.edu (from the most preferred to the least preferred). The algorithms will be distributed in the "first come first serve" order.
- Get to know how the algorithm works.
- Implement a serial version of the algorithm.
- Which parts of the algorithm allow parallelization?
- Choose your favorite parallel programming technology and implement a parallel version of the algorithm.
- Define and choose the best tuning parameters of the algorithm.
- Benchmark your implementation and compare with the results of the other groups.

2.4 Presentation

- The exact format will be defined later.