Leibniz-Universität Hannover
Institut für Kartographie & Geoinformatik
Prof. Dr. Martin Werner

Big Geospatial Data (Summer Term 2017)
Tutorial 2

Literatur: MPI tutorial, for example: *http://mpitutorial.com/tutorials/*
OpenMP tutorial, for example: *http://bisqwit.iki.fi/story/howto/openmp/*
MapReduce over MPI: *http://mapreduce.sandia.gov/doc/Interface_c++.html*

Aufgabe 4: Prime Numbers using the Message Passing Interface

Read some online tutorial about the message passing interface and try to solve the following problem. You can use your favourite MPI implementation, run our reference virtual machine (VM), or use our Docker image (not yet ready, we will finish it as soon as possible!) for both development and running. Note that you should assign more than one core to the VM in order to see parallelism pay off...
The example problem is to find all prime numbers below a given threshold. Two versions of the program shall be implemented: the first version just counts the number of primes and thus can use a simple Reduce operation to get the final count, the second version shall report all primes into a single array of integers at the root node.
Start off with the source code from Github *https://github.com/mwernerds/big_geospatial_data_-lecture/tree/master/04_mpi_prime* and perform the following:

– Read and understand `prime.cpp`, which calculates primes smaller than l in a simple for loop

– The MPI version shall be implemented starting from `mpi_prime.cpp`. We assume, there are N processors. The program shall find all prime numbers in $2 \dots l-1$. Extend the given file as follows:

  a. Uniformly assign a range of numbers to each processor based on the current rank: rank 0 shall get responsible for $2 \dots \frac{l}{N} - 1$, rank 1 for $\frac{l}{N} \dots 2 * \frac{l}{N} - 1$ and so on. Print out the range of numbers, each processor feels to be responsible for by changing the "Hello World" line.

  b. Use a loop in every processor to find all primes. Output the count.

  c. Use MPI_Reduce to collect all those counts into a single number and compare with the sequential implementation.

  d. You could actually try to get all the prime numbers into the main memory of the first processor. However, this needs a variant of MPI_Gather that accepts varying numbers of elements per node as our nodes will find different numbers of primes in their respective parts.
  The overall approach is then to first use MPI_Gather with the counts such that the target processor (typically rank 0) knows how much to expect from which other processor $(1 \dots N)$. Then, MPI_Gatherv provides a variant of Gather which allows to receive different sizes from different processors. The following code snippet works well, change it to fit your data structures:

```
// gather the individual counts
std::vector<int> nums(size);
MPI_Gather(&local_count,1,MPI_INT,&nums[0],1,MPI_INT,0,MPI_COMM_WORLD
    );
if (rank == 0)
  for (size_t i=0; i < size;i++)
    cout << i << ":"<< nums[i] << "\n" ;

std::vector<int> disps(size);
// Calculate storage locations for the resulting data
for (int i = 0; i < size; i++)
  disps[i] = (i > 0) ? (disps[i-1] + nums[i-1]) : 0;

// gather all prime numbers (different count from each processor!).
MPI_Gatherv(
    &primes[0], local_count, MPI_INT, // local input data
    &result[0],&nums[0],&disps[0],MPI_INT, // master storage info
    0,MPI_COMM_WORLD); // master 0, COMM_WORLD
```

e. Run the programs with various parameter settings. You can use `time ./prime` as well as `time mpirun -n 8 ./prime.mpi` to get runtime information. Be sure to replace 8 with a good number (e.g., the number of threads of your CPU). For my laptop, the OpenMP version (i.e., prime.cpp compiled with `g++ -o prime -fopenmp -Ofast prime.cpp -std=c++11` was faster until a maximum of 10 million. For 100 million, the MPI version (`mpicxx -o prime.mpi -Ofast -std=c++11 mpi_prime.cpp`) was finally faster.

Aufgabe 5: WordCount using MapReduce over MPI (optional)

As MapReduce is a programming pattern, it has been implemented in various environments, not only Apache Hadoop. In this tutorial, we want to shortly look at the MR-MPI implementation, which provides MapReduce over MPI and can thus be used to write C++ / MPI-based MapReduce programs.
The aim of this tuturial is to learn to know one environment in which MapReduce can be done from C++ / Python as the interoperability of MapReduce over Hadoop with these environments is limited or complicated.

– Download, compile and install MR-MPI from *www.sandia.gov/ sjplimp/download.html* or from our lecture page. Therefore, you need to unzip the package, go to the `src` directory, copy the Makefile for use with MPI to this directory under the name Makefile and run `make`. This has already been prepared for the VM image from the web page.

– Read and understand the wordfreq.cpp example

– Compile it and apply it to the Gutenberg selection from the lecture page

Aufgabe 6: MapReduce in R and Python

he most important consideration for MapReduce is how you actually formulate your given problem as a sequence of MapReduce instantiations. In order to be able to try out this new paradigm, this tutorial gives complete environments for R and Python in which you can use MapReduce on your toy problems.

- Look at mapreduce.py and run it for various texts.
- Look at mapreduce.R and run it for various texts.
- Change one of those to perform prime number counting using MapReduce.