Exercise 3
Using MPI

The goal of this assignment is to gain some experience on how to write, compile and run programs that are executed in parallel on a cluster. You will solve the problems with MPI and MPI through Slurm. In the next exercise you will solve the same problems with Hadoop.

This exercise has no mandatory delivery.

Introduction

In directory "/share/dat351/input" you will find three files with numbers (i.e. doubles) that you will work with in this exercise.

In this exercise you will parallelize the following tasks:

- Find the count of numbers summed over all three files.
- Find the smallest number.
- Find the largest number.
- Sum all the numbers.

If your programs are working, they should produce the following results:

- Number of elements: 3000000.
- Largest number: 9999.980000
- Smallest number: -9999.990000
- Sum of all numbers: -14968467.316438

For all of the tasks of this exercise, you will be working on virtual machines set up with Rocky8. Machines are set up on the VMWare server, and are accessible via SSH. From your own computer, you must use SSH to access the computers. All machines can be reached on the lab E425, or through the computer eple.hib.no.

Task one

Three machines have been set up with *mpich*, an MPI implementation.

- 10.0.0.201 (**mpi1.dat351**)
- 10.0.0.202 (**mpi2.dat351**)
- 10.0.0.203 (**mpi3.dat351**)

You can submit your MPI programs from either of these computers. We have created 20 user-accounts for you, **student0** to **student19**. Use the same account as you used in the previous lab. The computers have the the same shared files system and password database as the computers of the first lab exercise. You log on to these computers with the same password as you have used before.

The computers are configured to use rsh for *mpich*. You must create a **.rhosts** file to allow rsh login without password between the computers. Check that **rsh** is working between the computers, and that your are not asked for a password.

Create a C/C++ MPI program for doing the calculations outlined in the introduction, and use *mpich* to run the program. You should run three instances of the program. Each instance should do calculations using one of the files only. The master process, i.e. the process with rank 0, should collect the preliminary results, do the final calculations and present the answers.

An MPI demonstration

This section details how to run a simple MPI program. In the below C program, *hello.c*, the process with rank 0 sends a message to all the other processes.

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main (int argc, char **argv) {
    char message[20];
    int i, rank, size, tag=99;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    if(rank == 0) {
        strcpy(message, "Hello, world");
        for (i=1; i<size; ++i) {
            MPI_Send(message,13,MPI_CHAR,i,tag,MPI_COMM_WORLD);
        }
    }
    else {
        MPI_Recv(message,20,MPI_CHAR,0,tag,MPI_COMM_WORLD,&status);
    }
    printf( "Message from process = %d: %.13s\n", rank,message);
    MPI_Finalize();
    return 0;
}
```

The above code is also accessible as “/share/dat351/cprog/hello.c”.

Create a directory “mpi” in your home folder and copy the source code to this directory. Compile the program using the command below.

```
mpicc -o hello hello.c
```

Before you can run the program, you must create a **machines** file.

```
cat > machines <<EOF
mpi1.dat351
mpi2.dat351
mpi3.dat351
EOF
```

The below command will run three instances of the program, one on each of the mpi computers.

```
mpiexec -f machines -n 3 ./hello
```

Sum numbers from file with C

The C program below, *sum.c*, demonstrates how to read and sum real numbers that are read from a file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[]) {
    FILE * fp;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;
    char filename[40] = "/share/dat351/input/file";
    int rank;

    if (argc <=1 ) {
        printf("Missing argument, an integer in the range 0 to 2.\n");
    } else {
        rank = atoi(argv[1]);

        // Need filename as "/share/dat351/input/file" + rank
        char numchar[2]; // Need space for rank and \0
        sprintf(numchar,"%d",rank); // Write rank into char array
        strcat(filename,numchar);  // Appending rank to "filename"

        fp = fopen(filename, "r");
        if (fp == NULL) {
            printf("Could not open file \"%s\".\n",filename);
        } else {
            printf("Will sum numbers from file \"%s\".\n",filename);
            double sum=0;
            while ((read = getline(&line, &len, fp)) != -1) {
                if (read > 0) sum += atof(line);
            }
            printf("Sum is %f.\n",sum);

            fclose(fp);
            if (line) free(line);
        }
    }

    return 0;
}
```

The above code is also accessible as "/share/dat351/cprog/sum.c". Compile the program using the command below.

```
cc sum.c -lm -o sum
```

Write a MPI program that solves the tasks outlined in the introduction. Then compile and run the program using the three mpi nodes.

Task two

For the second task, you should use Slurm to run your MPI programs.

You have been working with Slurm in a previous lab exercise. Our Slurm "cluster" consist of only three nodes:

- 10.0.0.239 (**slurmmaster.dat351**) – Slurm server and worker node.
- 10.0.0.240 (**slurmw1.dat351**) – Slurm worker node.
- 10.0.0.241 (**slurmw2.dat351**) – Slurm worker node.

Slurm can use different approches for running MPICH tasks. MPICH can e.g. use Slurm as process management systems, instead of e.g. Hydra. Then MPI jobs can be run with the `srun` command inside `sbatch` scripts. Slurm directly launches the tasks and performs initialization of communications. This is the preferred approach. This requires that MPICH is built for Slurm. The MPI jobs are then run with the `srun` command as a normal Slurm job.

Another approach is to let Slurm create the resource allocation for the job, and then let e.g. Hydra launch the tasks. These tasks are then initiated outside of Slurm's monitoring or control.

The same MPICH build can not support both Hydra and Slurm, i.e. the two approaches can not live on the same MPI cluster.

On the the MPI cluster with mpi1, mpi2 and mpi3, Hydra is used for process management. The Slurm cluster is also set up with MPICH. On this cluster MPICH is built with Slurm support.

Run all your MPI programs from task one using Slurm as job scheduler. Remember to set the "--nodes" switch to use all the Slurm nodes.

- Are there any differences in the time consumption?
- Discuss the benefits of using MPI together with a job scheduler? Do you see any disadvantages?

You will only run the MPI jobs through Slurm as LRM, but also HTCondor and TORQUE have MPI support.