Lab Guide 8

Multiprocess Parallelization with MPI

Objective:

- parallelize the execution of an algorithm in a distributed memory environment

Introduction

This lab session aims to apply the basic MPI communication concepts studied in the previous session to parallelize specific computations of a simple application.
Copy the `/share/cpar/PL08_Codigo` folder to your home directory in the SeARCH cluster.

Compile the program in the cluster frontend using the `mpic++ -O2 -o primes PrimeMain.cpp` command. Use the `sbatch primes_mpi.sh` command to run the application.
The `primes_mpi.sh` file should specify the required resources and should run the MPI application.
The following example requests three PUs and spawns three MPI processes:

```
[search7edu]$ cat primes_mpi.sh
#!/bin/bash
#SBATCH --time=1:00
#SBATCH --ntasks=3
#SBATCH --partition=cpar
mpirun -np 3 ./primes
```

The number of requested resources (`--ntasks`) must be the same as the number of processes (`-np`) used in the `mpirun` command.

Exercise 1 – Prime calculation using the Sieve of Eratosthenes

Consider the following sequential program, which finds all prime numbers up to a given `MAXP`:

```
int MAXP = 1000000;
int SMAXP = 1000;
int pack=MAXP/10;

PrimeServer *ps1 = new PrimeServer();
PrimeServer *ps2 = new PrimeServer();
PrimeServer *ps3 = new PrimeServer();

ps1->minitFilter(1,SMAXP/3,SMAXP);
ps2->minitFilter(SMAXP/3+1,2*SMAXP/3,SMAXP);
ps3->minitFilter(2*SMAXP/3+1,SMAXP,SMAXP);

int *ar = new int[pack/2];
for(int i=0; i<10; i++) {
    generate(i*pack, (i+1)*pack, ar);
    ps1->mprocess(ar,pack/2);
    ps2->mprocess(ar,pack/2);
    ps3->mprocess(ar,pack/2);
}
ps3->end();
```

a) Parallelize the code using MPI through the implementation of a pipeline of processes that receives an array of integers, created by the `generate` function, and each process filters out a subset of the input. The `mprocess` method implements the filtering of the primes, and `end` prints the final amount of primes found. This pipeline should have 3 processes, one for each instance of `PrimeServer` performing the filtering.

b) Modify the parallelization implemented in **a)** to work with an arbitrary number of processes and messages.

c) Parallelize the sequential application through the implementation of a farm of processes behaving in a "work sharing" paradigm with dynamic scheduling.