INF3201- Parallel Programming

Assignment 1 - MPI

7 September - 26 September 2022

1 Parallel CrackMe challenge

A crackme is a challenge to evaluate the retro engineering skills of programmer. Here, it will be a challenge to evaluate your MPI parallelization skills, on a very simple task. You will try to find the value of a *char** created and expected by a function in our code, by covering all possibilities.

```
1 #include "crackme.h"
2
3 int main(int argc, char *argv[]){
4   int size = 30;
5   printf("%d\n", p(size, "I think this is the char* !"));
6
7   return 0;
8 }
```

Listing 1: A very simple call to function p: main.c

```
1     gcc main.c crackme.o -o testExecutable
```

The function p generates a *char** of a given size (first argument) and tests if the *char ** given by you (second argument) is equal to the generated one. You are not able to see what is generated by p. It only returns 0 when the two *char ** are equal, 1 otherwise. It is provided by the *crackme.o* object file, described in "crackme.h". Listing 1 shows one try to test if the value of the *char ** created by p is equal to "I think this is the char* !".

As it is not the correct one, *printf* line 5 outputs *1* on stdout.

2 Task

You first have to automate the generation of the *char ** in a sequential way. You then have to implement parallel version(s) that find the correct *char **, using MPI.

Hint: In this assignment, we expect and use *char ** from *C* language. Nowadays, in most systems, a *char* has a numerical value between -128 and 127 (both included). A *char* is a 1-byte value, i.e 8 bits, thus 255 possible values.

Reminder: (just in case) A *char** is a pointer that can be used to point to the first element of an array of char...

To evaluate the performance of your implementations, you need to run a set of measurements. We will focus on time measurements. The sequential version will be a baseline for comparison against your parallel version(s). Your solution should be able to scale when *size* and the number of processes varies.

3 Requirements

There are many ways to distribute the workload among the processes with such a problem. Choices made for this distribution will impact the scalability and performance of your solution. You should be able to explain why you decided to go with the chosen distribution(s).

3.1 General

- A high vigilance is given to plagiarism. Always reference when you are using resources. Don't use other's code.
- Your solution can not use any shortcuts that reduce the functionality of the program.
- Write code using C/C++ and MPI.
- Make sure the code is well-commented.
- We are trying to go as big as possible for the *size* of $char*$.

3.2 Time performance analysis

Evaluate the time performance of your parallel version. For that, you should study two dimensions: the number of processes, the size of input.

Have a graphical representation of your two studies (number of processes and size of input on the x axis and time to solution on the y axis). These two separate graphs help you answer the following questions: how well does your solution(s) scale, according to the number of processes (for the highest studied input size) ? How well does your solution(s) scale according to the size of the input (for a fixed number of processes)?

3.3 Report

The report should have the following sections:

- *Introduction* - describe your understanding of the assignment
- *Sequential solution* - explain your sequential solution
- *Parallel design solution* - describe how you parallelized the code, how the workload distribution is being done

- *Time performance analysis* - describe your experiments and results with different problem size and number of processes
- *Discussion* - discuss positive and negative points of your solution. Compare the theoretical maximum speedup with your results with different sizes while using different number of processes starting from 2 processes to the maximum available number of processes in the cluster.
- *Summary* - sums up and concludes your work

The report should be between 6 and 10 pages. Remember to:

- Explain everything in detail and answer questions from the assignment.
- Make your figures clear and understandable. Include references, captions and axes descriptions.
- Take into consideration the hardware used (e.g heterogeneity).

4 Archive

In this archive you can find a directory called *code* that contains:

- *crackme.h* - Header file of the crackme. Needs to be included to use p.
- *crackme.o* - Object file that contains a compiled version of the crackme. Needs to be linked at compilation to use p.
- *solution.txt* - file to store the numerical values found for p's *char**, one per size
- *hostfile* - input for *mpiexec*. Represents the list of hosts to launch MPI processes.
- *generate_hosts.sh* - Helps you generate a hostfile. Used in run.sh
- *mainMPI.c* - A very simple main calling p, inside an MPI environment.
- *dirtyMake.sh* - "-make" option proposes a dirty way to compile mainMPI.c
- *run.sh* - Another helper that executes *generate_hosts.sh* and runs a compiled version of mainMPI.c using the generated $hostfile$. For example:

```
1    ./run.sh 8 2 mainMPI 10
```

runs $mainMPI$ with a size of 10, asks for 8 processes, launched on two hosts, located in $hostfile$.

5 uvcluster

The Computer Science department has a cluster of nodes that can be used to run your solution. The cluster has a distributed file system. You need to copy your files only to the frontend (uvcluster.cs.uit.no) to access them from all other nodes.

In order to login to the cluster, use (in linux):

```
ssh <your UiT ID>@uvcluster.cs.uit.no
```

Make sure that you are familiar with the welcome message from the cluster, helping you use the cluster correctly.

6 Hand-in

You will work alone for this assignment. GitHub classroom is used as a hand-in platform for the course. You can and probably should commit and push as often as possible.

Remember to push all your changes before Sep 26 September 2022 23:59:59. Any changes pushed to your repository after that deadline will not be evaluated.