Université Libre de Bruxelles

Real-Time Operating Systems

Mastermind solver

Authors:
ABOU ZAIDI, Ahmed
AZZOUZI, Samia
SEFU Kevin

December 14, 2018

1 Introduction

In this second project we had to implement a parallel Mastermind solver using MPI on HYDRA. The main goal of this project was to help us become familiar with parallel algorithms and to understand the benefits and drawbacks that parallel algorithm present.

2 Implementation choices

2.1 Division of the task between processes

To understand how we decide to split the solving task between the processes, let consider the following example :
Let consider a mastermind game with S spots and N colors.

Step 1 : Calculate the number of all possible guesses that could be generated:

$$\frac{N!}{(N-S)!}$$

Step 2 : Divide the number return in Step 1 by the number of processes -1 (we don't take into account the master node). The step 2 will return for all processes the maximum number of guesses that the process will generate in the worst case and the index from which this generation will start. Lets call those two numbers G and I

Step 3 : At each turn all the processes will send (starting from I), the next plausible guess until the process reach the number G.

T he following graphics explain those 3 steps :

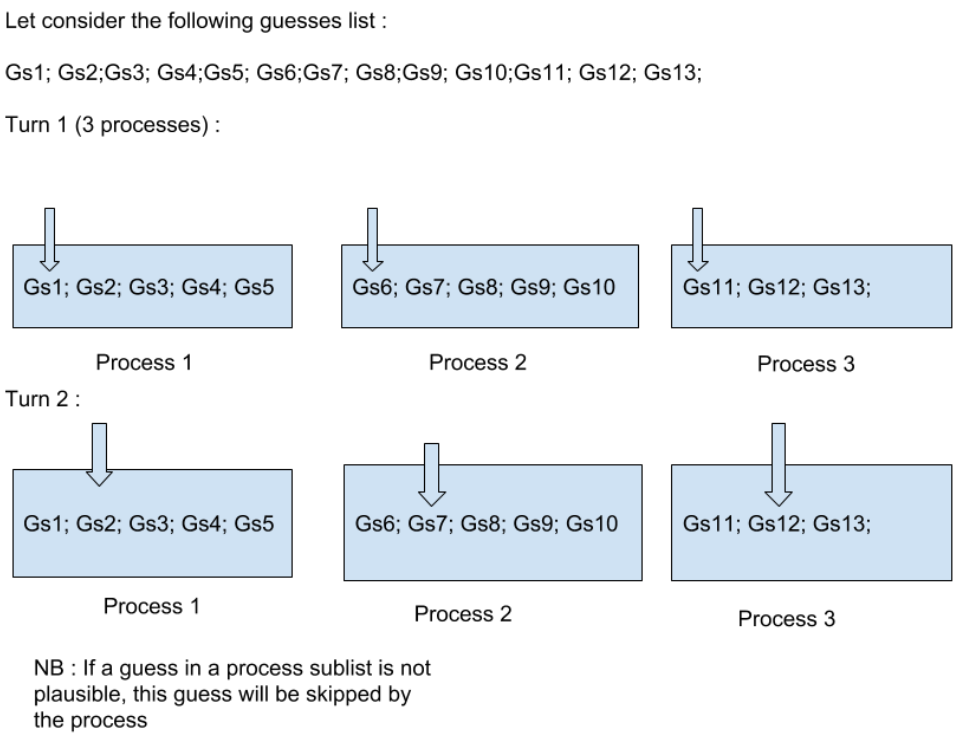

Figure 1: Maximum number of colors

2.2 Criteria for comparing two evaluation

An evaluation is a pair of two integers: $perfect$ and $colourOnly$. To determine the best between two evaluation we take the one with a higher number of $perfect$. If $perfect$s are equals we take the higher $colourOnly$.

2.3 Choice of int $size\_t$ instead of Int

To represent colors, we decided to not use interger data type but to use int $size\_t$ because int $size\_t$ provide more colors choices than Int.
The following graph shows the maximum numbers of possible colors (with the worst case where the number of spots is equal to the number of colors) using int $size\_t$:

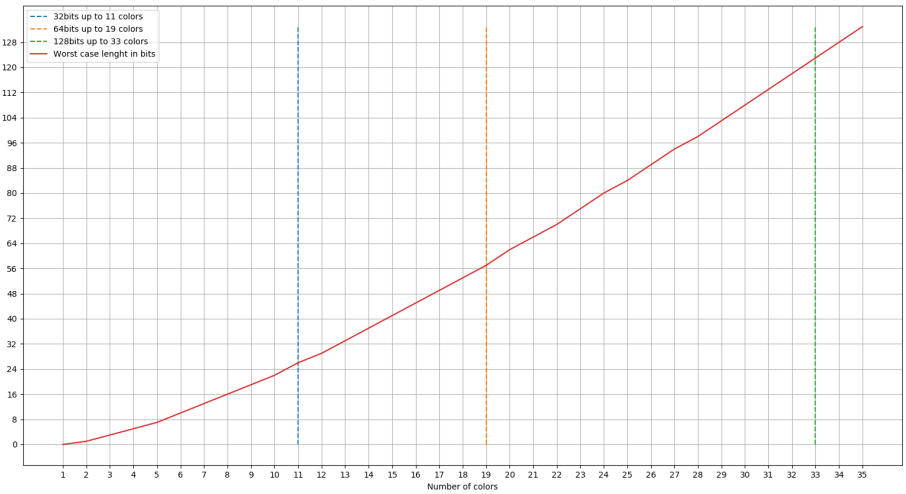

Figure 2: Maximum number of colors

3 MPI Protocal used

In our project we decide to us two MPI protocols, firstly the MPI Gather and secondly the MPI BCast.
We used the MPI Gather in order to get from the game master all the guess that players node generate therefore the master's receive buffer size at each turn was equal to the spot's number * the number of process.

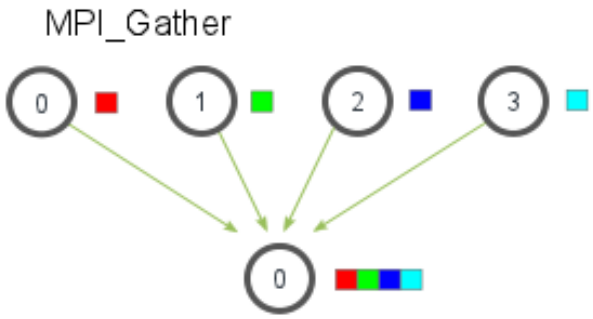

Figure 3: MPI Gather

Secondly we used MPI BCast in order to send a guess and its evaluation to all player's node.

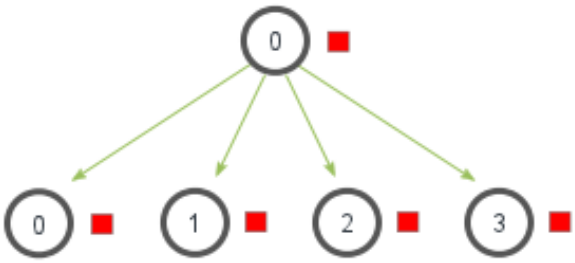

Figure 4: MPI BCast

4 Class diagram

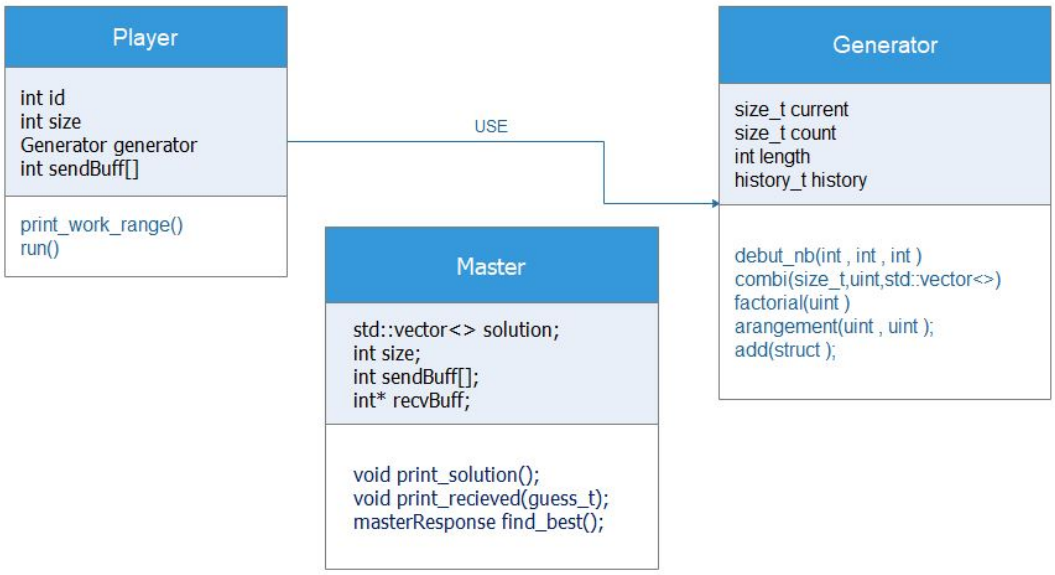

Figure 5: Class diagram

Table 1: Time in seconds for respectively 4 and 8 processors

S \ C	**10**	**12**	**14**
4	0,3	0,38	0,4
6	1,05	4,6	9,95
8	9,63	154,4	827

S \ C	**10**	**12**	**14**
4	0,29	0,57	0,6
6	1,25	1,99	9,61
8	10,81	128,32	787,95

5 Performances and Limitations

About our code performances and limitations we observed after several running that, the solution was found in at most 6 turn for a spot range of 6 and color range of 15. We also observed that our algorithm end in at most 25 seconds for those range also.

As we will see in the next section when the number of color is high and the number of spots is too close to the number of colors.

In table 1 we can see some running time examples.

6 Formula giving the number of possible guesses at start depending on spots and colors number

Let call S the spots number, C the colors number and P the set of plausible guesses at each turn.
The following formula give the number of possible guesses at start depending on spots and colors number :

$$\frac{C!}{(C-S)!}$$

7 Difficulties during project implementation

7.1 Plausible Guess

During the project a big difficulty that we met was to understand, the concept of plausible guess and how to implement it in our source code. To overcome this problem we decide to download on our personal cell phones the mastermind game and to try to understand this notion.

To know if a guess is plausible we should do the same process the master do with the solution. We evaluate olds guess and take the current geuss as solution. If we don't encounter any contradiction (different number of $perfect$ or different number of $colourOnly$) we know that the current guess is plausible. Otherwise we continue the brute force.

7.2 Hydra message protocol

As hydra was something new to us, it wasn't something easy for us to understand directly how hydra's message protocols works. This problem were solved by simply reading the file available on UV and by practicing.

8 Personal critics on parallel algorithm

We discover that parallelization was a powerfull method because it allows us to split a task into several nodes and therefore to speed up the time needed for solving the initial problem but we observed also that parallelization has costs such as :

1. Sometimes not all processors are used, this can happen because of the intrinsic sequential nature of a given step or part of the problem.
2. Parallel algorithm are harder to design
3. Delays of the communication

When a processors has no more plausible guess in its burte force range it should consume its range to figure it out. While other processors are ready to send their guesses.

And even worse after consuming its range the processor become useless.

9 Conclusion

As said in the introduction, we built multi-processes algorithm for solving the mastermind game. This project helped us to understand deeply how multi-processing and parallelization works. The project allowed us also to see, how powerful can be this method but it also revealed us limitations of parallelization.

10 Sources

Google images :

1. MPI Gather Image.
2. MPI BCast Image

11 Running the code

A `make run` command will compile and run the program.

The number of colors and spots are defined as preprocessor macros in the `mastermind.hpp` file where they can be changed.

Another way to set them is as parameters to the compiler (for example `-DCOLORS=10 -DSPOTS=4`).