RTOS-2

INFO-F404 Mastermind Solver

Christopher Lample, Ali Dhanani, Samineh Sadat Naghavi

1. Introduction

For this project, we implemented a solver for the Mastermind game. The solver is capable of running with an arbitrary number of computing nodes using the library MPI. Special care was taken to evenly divide work among the computing nodes and to minimize idle processor time.

2. Program execution

The program can be run on HYDRA with the following commands:

$ module load Boost/1.71.0-gompi-2019b
$ module load OpenMPI/3.1.4-GCC-8.3.0
$ make
$ mpirun -np <number of processors> mastermind <number of spaces> <number of colors>

The number of spaces and colors are optional arguments. If they are not specified, a default of four spaces and ten colors will be used.

3. Implementation

The mastermind solver is divided into two main components: the Guesser class and the GameMaster class. One process is dedicated to the GameMaster, and each other process is dedicated to a single Guesser instance. The Guesser instances search for a plausible guess, and report plausible guesses to the GameMaster. The GameMaster then evaluates the guess and distributes the response among all Guesser instances.

3.1 Division of solution space

A primary issue in implementing the mastermind solver was deciding how to allocate work among the Guesser instances. To resolve this, we consider each color to be a number in 0 through $n-1$, where n is the number of colors. Each possible color sequence in the solution space is then represented by a number in base n with s digits, where s is the number of spaces. This means that the solution space has size n^s - the same as the amount of numbers representable with s digits in base n.

By considering the possible guesses as numbers, we then divided the solution space evenly among the k Guesser instances in the following way. The first guesser instance begins processing with color sequence 0, and obtains the next color sequence by incrementing the previous color sequence by k. Similarly, the second guesser instance begins with color sequence 1, and increments in steps of k. The n^{th} node begins with color sequence n, and also increment in steps of k. In this way, each color sequence c belonged to exactly one guesser instance, namely the instance with an id of $c \bmod k$.

3.2 Avoiding race conditions

Since the Guesser instances all run in parallel, we needed to take special care to avoid race conditions. In particular, the solver should only make plausible guesses - guesses which do not not contradict *any* previous guess. If two Guesser instances submit guesses to the GameMaster at the same time, the GameMaster cannot simply evaluate them both. If it did this, one guess may make the other implausible and the program would be submitting an implausible guess.

To resolve this issue, Guesser nodes include a *guess number* along with their proposed guess. This indicates the number of previous guesses that are compared with the proposed guess. The GameMaster can then compare this guess number against its own count of the previous guesses, to detect any invalid guesses arising from race conditions. In particular, Guesser nodes will send an instance of the ProposedGuess class when reporting a plausible guess to the GameMaster. In response, the GameMaster will send an instance of the RespondedGuess class to all nodes. Inter-process communication occurs using MPI point-to-point communication.

3.3 Libraries

In implementing the solver, we made use of a testing library named Catch as well as the Boost MPI library. The Catch library allowed us to write unit tests and helped us detect several bugs while implementing the solver. The tests are automatically run when building the project.

The Boost MPI library provided C++ bindings for MPI. We believe this made the code significantly more concise compared to directly using the MPI C library. One key advantage of using the Boost library was its support for serialization. The library made it relatively simple to serialize the ProposedGuess and RespondedGuess classes which we send between the GameMaster and Guesser instances.

3.4 Difficulties and Limitations

As discussed earlier, the solution space for s spaces and n colors is n^s. This solution space grows quite rapidly, for example reaching a billion possible solutions when using only 10 colors and 9 spaces. In searching for an algorithm for the solver, we spent a significant amount of time looking for an algorithm which would not need to check each particular solution for plausibility. Although, we were unable to find such an algorithm, and our program checks each possible solution for plausibility.

It would be interesting to either prove that this is an optimal solution, or to construct a faster algorithm.

4. Conclusion

While the Mastermind game appears simple on the surface, the complexity quickly grows with the number of spaces and colors. For this project, we were able to develop a solver which can run in parallel and scale to an arbitrary number of processors. Our approach will check each possible solution for plausibility, until the actual solution has been guessed. This approach performs well for relatively low number of colors and spaces, but a different approach will be needed with an increased number of colors and spaces.