HPC4DS Project - Sudoku Puzzle with MPI

Course Name: High-Performance Computing for Data Science
Team Members: Jing Yan (jing.yan@studenti.unitn.it)
Department of Information Engineering and Computer Science (DISI)
The University of Trento, 2021-2022

Updated: Dec 11, 2021

1. Introduction

1.1 Challenge, State of the art, and Motivation

Basically, there are two significant challenges during the project.

The first one is how to apply parallel computing to the Sudoku puzzle. In the beginning, I implemented the serial code for the Sudoku puzzle, in which the main idea is to traverse all the possible solutions until figuring out a valid one. As the serial program will run in a sequence when traversing all the numbers (1-9) in each blank cell of the Sudoku map one by one, in this case, one process to run the code will be enough. However, when the user does not fill in enough cells of the sudoku puzzle when executing the serial program, there could be more than one solution to the Sudoku puzzle. I got inspired by the possible multiple solutions to a Sudoku puzzle and tried to combine the Sudoku puzzle with MPI for the first time. The first version code can be seen at this link. The problem is, even the parallel computing concept is introduced to the code through multiple processes in MPI, indeed, one process still works on one solution and outputs the solution separately, which is not expected after confirming with the professor. Then I tried the second version. The main idea is to calculate the total number of solutions to a sudoku puzzle through the cooperation of multiple processes.

The second one is the implementation of the sudoku puzzle with MPI, namely how to divide the computing tasks of solving the sudoku puzzle and assign the sub-tasks to multiple processes. Detailed info regarding the implementation will be introduced in the third chapter. After solving the issue, currently, the program could support the number of processes ranging from 1 to 729, which is the strong point of this program. (However, it seems that HPC has some restrictions regarding setting the number of processes. An error will be raised in the .e file when setting too many processes, for example, 500.)

Regarding the shortcomings of the MPI code, first, the user should make sure there will be multiple solutions to the Sudoku puzzle when executing the program (The program could also solve the Sudoku puzzle with only 1 solution, but it makes no sense to my MPI code.), second, I did not implement the error handler code to deal with the possible wrong input from the user when executing the program, third, the expected result of this program should be, when solving the same sudoku puzzle, the more processes to be set when submitting the job in HPC, the time cost should be less. However, the results I got when running the program with different Sudoku puzzles and different numbers of processes do not follow the previous theory strictly (I will explain this in the final discussion part, as far as I can see, it is reasonable, cause when one step is wrong when filling in the sudoku puzzle, no matter what you do next, all the effort will be in vain, plus, more processes means more communications.).

As for the motivation to choose this topic, I was so interested in Sudoku puzzles a few years ago. After spending more than 300 hours on this game, I could solve Sudoku puzzles with an expert difficulty level within approximately 10 minutes or even less sometimes. Thus, the previous experience with the Sudoku puzzle motivates me to figure out the solutions with code.

2. Problem Analysis

2.1 Serial Code

The source code “sudoku_serial.c” can be seen in the attachment, and the following snapshot is an example of execution.

- When commenting on the code to print the Sudoku puzzle

```
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 4 6 1 8 5 2 2 8 2
4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1
7 9 3 8 2 4 8 4 3 8 5 1 9 2 2 9 9 8
There are 1 solutions for the sudoku puzzle!
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 4 6 1 8 5 2 2 8 2
4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1
7 9 3 8 2 4 8 4 3 8 5 1 9 2 2
There are 55 solutions for the sudoku puzzle!
```

- When uncommenting on the code to print the Sudoku puzzle

```
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 4 6 1 8 5 2 2 8 2
4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1
7 9 3 8 2 4 8 4 3 8 5 1 9 2 2 9 9 8
---------|---------|---------|
 2       | 6       |    5    |
    8    | 9     2 |         |
         |    4    | 6       |
---------|---------|---------|
         | 4     9 |       6 |
 5       |         |    9    |
    9    | 2       | 1  4    |
---------|---------|---------|
       1 |         |       3 |
    4    | 3  1    |         |
    2    |         |       8 |
---------|---------|---------|

---------|---------|---------|
 2  1  9 | 6  7  3 | 8  5  4 |
 4  8  6 | 9  5  2 | 7  3  1 |
 7  3  5 | 1  4  8 | 6  2  9 |
---------|---------|---------|
 1  7  2 | 4  3  9 | 5  8  6 |
 5  6  4 | 7  8  1 | 3  9  2 |
 8  9  3 | 2  6  5 | 1  4  7 |
---------|---------|---------|
 6  5  1 | 8  2  4 | 9  7  3 |
 9  4  8 | 3  1  7 | 2  6  5 |
 3  2  7 | 5  9  6 | 4  1  8 |
---------|---------|---------|

There are 1 solutions for the sudoku puzzle!
```

2.2 Workflow of the Serial Code

2.2.1 Obtain the Sudoku puzzle from the user

The function “int ParseArgv(int argc, char **argv, char map[])” will generate the Sudoku map based on the user’s input when executing the program. As the two snapshots in 2.1 show, the program will accept the parameter with the following format: ./sudoku_serial cell_m_x-axis cell_m_y-axis cell_m_value cell_n_x-axis cell_n_y-axis cell_n_value ……

The 81 cells in the sudoku map will be traversed through coordinates which range from (1,1) to (9,9) row by row, and the value in each cell will be read/written from/to from map[0] to map[80]. The following is an example of how to obtain the Sudoku puzzle from the user. (The blank cells will be filled in 0 through this function, but 0 won't be displayed when printing the Sudoku puzzle.)

```
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 4 6 1 8 5 2 2 8 2
4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1
7 9 3 8 2 4 8 4 3 8 5 1 9 2 2 9 9 8
---------|---------|---------|
 2       | 6       |    5    |
    8    | 9     2 |         |
         |    4    | 6       |
---------|---------|---------|
         | 4     9 |       6 |
 5       |         |    9    |
    9    | 2       | 1  4    |
---------|---------|---------|
       1 |         |       3 |
    4    | 3  1    |         |
    2    |         |       8 |
---------|---------|---------|
```

2.2.2 Solve the Sudoku Puzzle

The function “int SudokuSolution(char map[])” will calculate the valid solutions and the number of valid solutions to the Sudoku puzzle which is generated from 2.3.1 based on the input of the user.

First, all the blank cells will be stored in “int as[81][2]” (The value of each blank cell should be 0 and there will be 81 blank cells at the maximum), the number of the blank cells is stored in “step” variable, thus the task should be trying to fill the valid numbers in the blank cells.

Second, traverse from 1 to 9 in order and try to fill a number in blank cells one by one. If there is a number from 1 to 9 (namely the current number, recorded in “cur”) in the current blank cell (pos) that makes the current Sudoku puzzle work, repeat the same step to the next blank cell (pos+1). Otherwise, the previous steps might go wrong which caused that the program could not find a valid number for the current blank cell, then go back to the last blank cell (it was filled in with a cur), and try cur + 1 for the last blank cell. In summary, try all the numbers from 1 to 9 in each blank cell. If there is a number in the current blank cell that makes the current Sudoku puzzle work, repeat the steps for the next blank cell, otherwise, go back to the last blank cell and try the next value. Repeating until the Sudoku map is finished. When pos equals step (finishing traversing all the blank cells), it means the valid solution is found.

During the second step, the function “int IsValid(int x, int y, int value, char map[])” is used to check whether the value in the coordinates (x,y) of the sudoku map is valid. In which, there are three judgments regarding whether the value is duplicated in row x, or column y (which are finished in the function “int RowColumnCheckDup(int x, int y, char value, char map[])”), or the box (which is finished in the function “int BoxCheckDup(int x, int y, int value, char map[])”). (The Sudoku map is divided into 9 small boxes).

2.3 Benchmark of the Serial Code on my own laptop

```
yanjing@yanjingdeMacBook-Pro sudo_mpi % gcc sudoku_serial.c -o sudoku_serial
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 2 7 1 4 6 1 8 5
 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1
6 8 4 7 3 1 7 9 3 8 2 4
The num of solutions is 3258, total time is 310 ms.
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 2 7 1 4 6 1 8 5
 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1
6 8 4 7 3 1 7 9 3
The num of solutions is 146106, total time is 4173 ms.
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 2 7 1 8 5 2 2 8
 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4
7 3 1 7 9 3
The num of solutions is 721288, total time is 19630 ms.
yanjing@yanjingdeMacBook-Pro sudo_mpi % ./sudoku_serial 1 1 2 1 2 7 1 4 6 1 8 5
 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1
6 8 4 7 3 1
The num of solutions is 1912148, total time is 48072 ms.
```

3. Main Steps towards Parallelization

3.1 Design of the Parallel Solution

First of all, all the processes set in the PBS file will be divided into two categories. Process 0 (namely my_rank = 0, which is called master in the program) will take responsibility for adding all the calculation results from other processes as a receiver. Other processes (which are called slave processes) will calculate the possible solutions to the Sudoku puzzle and send the number of solutions to process 0 separately.

Regarding how to make slave processes execute different sub-tasks separately, I introduced the following idea, namely a variable “level”, which could be equal to 1, 2, or 3 based on the value of comm_sz in PBS file.

- When the value of comm_sz belongs to [3, 10], which means the number of slave processes should belong to [2, 9], the variable “level” will be set as 1.
- When the value of comm_sz belongs to [11, 82], which means the number of slave processes should belong to [10, 81], the variable “level” will be set as 2.
- When the value of comm_sz belongs to [83, 730], which means the number of slave processes should belong to [82, 729], the variable “level” will be set as 3.

At the same time, another variable “taskNum” will be introduced. The value of the variable “taskNum” is based on the value of the variable “level”.

- When “level” equals 1, the “taskNum” will be set as 9. Here, the 9 tasks will be filling in the first blank cell in the Sudoku puzzle with [1, 9], which will assign to the slave processes evenly (the number of slave processes should belong to [2, 9]) will be assigned. After filling in the first blank cell with 1 number, each slave process will call the “SudokuSolution” function to calculate the number of solutions. If the first blank cell is filled in with an invalid number, the slave process will return 0 as the solutions are invalid.
- When “level” equals 2, the “taskNum” will be set as 81. Here, the 81 tasks will be filling in the first two blank cells in the Sudoku puzzle with [1, 9], which will assign to the slave processes evenly (the number of slave processes should belong to [10, 81]) will be assigned. After filling in the first blank cells with 2 numbers, each slave process will call the “SudokuSolution” function to calculate the number of solutions. If the first two blank cells are filled in with invalid numbers, the slave process will return 0 as the solutions are invalid.
- When “level” equals 3, the “taskNum” will be set as 729. Here, the 729 tasks will be filling in the first three blank cells in the Sudoku puzzle with [1, 9], which will assign to the slave processes evenly (the number of slave processes should belong to [82, 729]) will be assigned. After filling in the first three blank cells with 3 numbers, each slave process will call the “SudokuSolution” function to calculate the number of solutions. If the first three blank cells are filled in with invalid numbers, the slave process will return 0 as the solutions are invalid.

During the previous step, regarding how to fill in the first blank cell, the first two blank cells or the first three blank cells, three for loops from line 143 to line 161 in “mpi_parallel.c” will generate the numbers and store the numbers in the variable param[3] of the struct workInfo.

3.2 Implementation

The source code “sudoku_parallel.c”, “sudoku_parallel.h”, and “mpi_parallel.c” can be seen in the attachment, and the following snapshot is an example of execution.

```
[jing.yan@hpc-head-n2 ~]$ module load mpich-3.2
[jing.yan@hpc-head-n2 ~]$ mpicc -std=c99 -g -Wall -o mpi_parallel mpi_parallel.c sudoku_parallel.c
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh
#!/bin/bash
#PBS -l select=10:ncpus=2:mem=2gb
#PBS -l walltime=0:05:00
#PBS -q short_cpuQ
module load mpich-3.2
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5
8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1 7 9 3 8 2 4
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5
8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1 7 9 3 8 2 4
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5
8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1 7 9 3
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5
8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1 7 9 3
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6
2 9 6 4 2 6 7 1 6 8 4 7 3 1 7 9 3
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6
2 9 6 4 2 6 7 1 6 8 4 7 3 1 7 9 3
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5
8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1
mpiexec -n 3 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5
8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1
[jing.yan@hpc-head-n2 ~]$ qsub sudo.sh
8205092.hpc-head-n1.unitn.it
```

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205092
The num of processes is 2, the num of solutions is 3258, total time is 196 ms.
The num of processes is 2, the num of solutions is 3258, total time is 204 ms.
The num of processes is 2, the num of solutions is 146106, total time is 3095 ms.
The num of processes is 2, the num of solutions is 146106, total time is 3141 ms.
The num of processes is 2, the num of solutions is 721288, total time is 13926 ms.
The num of processes is 2, the num of solutions is 721288, total time is 13406 ms.
The num of processes is 2, the num of solutions is 1912148, total time is 30606 ms.
The num of processes is 2, the num of solutions is 1912148, total time is 30592 ms.
```

3.3 Benchmark on the HPC@UniTrento Cluster

Similar to the execution examples shown in the snapshots in the previous chapter, I tried more scenarios and got the performance info as follows.

// #PBS -l select=10:ncpus=2:mem=2gb

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205092
The num of processes is 2, the num of solutions is 3258, total time is 196 ms.
The num of processes is 2, the num of solutions is 3258, total time is 204 ms.
The num of processes is 2, the num of solutions is 146106, total time is 3095 ms.
The num of processes is 2, the num of solutions is 146106, total time is 3141 ms.
The num of processes is 2, the num of solutions is 721288, total time is 13926 ms.
The num of processes is 2, the num of solutions is 721288, total time is 13406 ms.
The num of processes is 2, the num of solutions is 1912148, total time is 30606 ms.
The num of processes is 2, the num of solutions is 1912148, total time is 30592 ms.
```

// #PBS -l select=10:ncpus=2:mem=2gb

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205113
The num of processes is 8, the num of solutions is 3258, total time is 172 ms.
The num of processes is 8, the num of solutions is 3258, total time is 172 ms.
The num of processes is 8, the num of solutions is 146106, total time is 3458 ms.
The num of processes is 8, the num of solutions is 146106, total time is 3472 ms.
The num of processes is 8, the num of solutions is 721288, total time is 11368 ms.
The num of processes is 8, the num of solutions is 721288, total time is 11788 ms.
The num of processes is 8, the num of solutions is 1912148, total time is 37654 ms.
The num of processes is 8, the num of solutions is 1912148, total time is 36373 ms.
```

// #PBS -l select=10:ncpus=2:mem=2gb

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205120
The num of processes is 16, the num of solutions is 3258, total time is 123 ms.
The num of processes is 16, the num of solutions is 3258, total time is 89 ms.
The num of processes is 16, the num of solutions is 146106, total time is 1478 ms.
The num of processes is 16, the num of solutions is 146106, total time is 1453 ms.
The num of processes is 16, the num of solutions is 721288, total time is 6489 ms.
The num of processes is 16, the num of solutions is 721288, total time is 5860 ms.
The num of processes is 16, the num of solutions is 1912148, total time is 16834 ms.
The num of processes is 16, the num of solutions is 1912148, total time is 17369 ms.
```

// #PBS -l select=40:ncpus=2:mem=2gb

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205121
The num of processes is 32, the num of solutions is 3258, total time is 98 ms.
The num of processes is 32, the num of solutions is 3258, total time is 89 ms.
The num of processes is 32, the num of solutions is 146106, total time is 1482 ms.
The num of processes is 32, the num of solutions is 146106, total time is 1460 ms.
The num of processes is 32, the num of solutions is 721288, total time is 5107 ms.
The num of processes is 32, the num of solutions is 721288, total time is 4981 ms.
The num of processes is 32, the num of solutions is 1912148, total time is 17418 ms.
The num of processes is 32, the num of solutions is 1912148, total time is 16905 ms.
```

// #PBS -l select=40:ncpus=2:mem=2gb

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205124
The num of processes is 64, the num of solutions is 3258, total time is 101 ms.
The num of processes is 64, the num of solutions is 3258, total time is 84 ms.
The num of processes is 64, the num of solutions is 146106, total time is 1790 ms.
The num of processes is 64, the num of solutions is 146106, total time is 1835 ms.
The num of processes is 64, the num of solutions is 721288, total time is 4264 ms.
The num of processes is 64, the num of solutions is 721288, total time is 4237 ms.
The num of processes is 64, the num of solutions is 1912148, total time is 18194 ms.
The num of processes is 64, the num of solutions is 1912148, total time is 19071 ms.
```

// #PBS -l select=70:ncpus=2:mem=2gb

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205126
The num of processes is 128, the num of solutions is 3258, total time is 84 ms.
The num of processes is 128, the num of solutions is 3258, total time is 75 ms.
The num of processes is 128, the num of solutions is 146106, total time is 1088 ms.
The num of processes is 128, the num of solutions is 146106, total time is 1069 ms.
The num of processes is 128, the num of solutions is 721288, total time is 2367 ms.
The num of processes is 128, the num of solutions is 721288, total time is 2376 ms.
The num of processes is 128, the num of solutions is 1912148, total time is 12847 ms.
The num of processes is 128, the num of solutions is 1912148, total time is 12540 ms.
```

4. Final Discussion

Basically, I can see from the performance results above, the more processes, the less time cost. When the number of solutions to the Sudoku puzzle is around 800000, for example, 721288, the result is following the expected behavior strictly.

```
[jing.yan@hpc-head-n2 ~]$ cat result_log | grep 721288
The num of processes is 2, the num of solutions is 721288, total time is 13926 ms.
The num of processes is 2, the num of solutions is 721288, total time is 13406 ms.
The num of processes is 8, the num of solutions is 721288, total time is 11368 ms.
The num of processes is 8, the num of solutions is 721288, total time is 11788 ms.
The num of processes is 16, the num of solutions is 721288, total time is 6489 ms.
The num of processes is 16, the num of solutions is 721288, total time is 5860 ms.
The num of processes is 32, the num of solutions is 721288, total time is 5107 ms.
The num of processes is 32, the num of solutions is 721288, total time is 4981 ms.
The num of processes is 64, the num of solutions is 721288, total time is 4264 ms.
The num of processes is 64, the num of solutions is 721288, total time is 4237 ms.
The num of processes is 128, the num of solutions is 721288, total time is 2367 ms.
The num of processes is 128, the num of solutions is 721288, total time is 2376 ms.
```

However, when the number of solutions to the Sudoku puzzle is 3258 or 146106, the trend for time cost to reduce when the number of processes is increasing is not strict. I think that is caused by the workload being too less for the multiple processes.

```
[jing.yan@hpc-head-n2 ~]$ cat result_log | grep 146106
The num of processes is 2, the num of solutions is 146106, total time is 3095 ms.
The num of processes is 2, the num of solutions is 146106, total time is 3141 ms.
The num of processes is 8, the num of solutions is 146106, total time is 3458 ms.
The num of processes is 8, the num of solutions is 146106, total time is 3472 ms.
The num of processes is 16, the num of solutions is 146106, total time is 1478 ms.
The num of processes is 16, the num of solutions is 146106, total time is 1453 ms.
The num of processes is 32, the num of solutions is 146106, total time is 1482 ms.
The num of processes is 32, the num of solutions is 146106, total time is 1460 ms.
The num of processes is 64, the num of solutions is 146106, total time is 1790 ms.
The num of processes is 64, the num of solutions is 146106, total time is 1835 ms.
The num of processes is 128, the num of solutions is 146106, total time is 1088 ms.
The num of processes is 128, the num of solutions is 146106, total time is 1069 ms.
[jing.yan@hpc-head-n2 ~]$
[jing.yan@hpc-head-n2 ~]$
[jing.yan@hpc-head-n2 ~]$ cat result_log | grep 3258
The num of processes is 2, the num of solutions is 3258, total time is 196 ms.
The num of processes is 2, the num of solutions is 3258, total time is 204 ms.
The num of processes is 8, the num of solutions is 3258, total time is 172 ms.
The num of processes is 8, the num of solutions is 3258, total time is 172 ms.
The num of processes is 16, the num of solutions is 3258, total time is 123 ms.
The num of processes is 16, the num of solutions is 3258, total time is 89 ms.
The num of processes is 32, the num of solutions is 3258, total time is 98 ms.
The num of processes is 32, the num of solutions is 3258, total time is 89 ms.
The num of processes is 64, the num of solutions is 3258, total time is 101 ms.
The num of processes is 64, the num of solutions is 3258, total time is 84 ms.
The num of processes is 128, the num of solutions is 3258, total time is 84 ms.
The num of processes is 128, the num of solutions is 3258, total time is 75 ms.
```

Another problem is that when the number of the solutions to the Sudoku puzzle is 1912148, I uncomment line 162 in “mpi_parallel.c” and get the following information. Actually, if I keep increasing the number of processes, it is possible that the time cost might increase too, as the number of valid slave processes (means the number of solutions returned by the slave process is not 0) won’t be too large.

```
[jing.yan@hpc-head-n2 ~]$ cat result_log | grep 1912148
The num of processes is 2, the num of solutions is 1912148, total time is 30606 ms.
The num of processes is 2, the num of solutions is 1912148, total time is 30592 ms.
The num of processes is 8, the num of solutions is 1912148, total time is 37654 ms.
The num of processes is 8, the num of solutions is 1912148, total time is 36373 ms.
The num of processes is 16, the num of solutions is 1912148, total time is 16834 ms.
The num of processes is 16, the num of solutions is 1912148, total time is 17369 ms.
The num of processes is 32, the num of solutions is 1912148, total time is 17418 ms.
The num of processes is 32, the num of solutions is 1912148, total time is 16905 ms.
The num of processes is 64, the num of solutions is 1912148, total time is 18194 ms.
The num of processes is 64, the num of solutions is 1912148, total time is 19071 ms.
The num of processes is 128, the num of solutions is 1912148, total time is 12847 ms.
The num of processes is 128, the num of solutions is 1912148, total time is 12540 ms.
```

// #PBS -l select=130:ncpus=2:mem=2gb
// mpiexec -n 257 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1
// mpiexec -n 257 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205133 | grep time
The num of processes is 256, the num of solutions is 1912148, total time is 15228 ms.
The num of processes is 256, the num of solutions is 1912148, total time is 15274 ms.
```

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205133 | grep "solutions is 0" | wc -l
492
```

Here, I set the same mpiexec twice, so the number of invalid slave processes (means the number of solutions returned by the slave process is 0) should be 246.

// #PBS -l select=20:ncpus=2:mem=2gb
// mpiexec -n 33 ./mpi_parallel 1 1 2 1 2 7 1 4 6 1 8 5 2 2 8 2 4 9 2 6 2 3 5 4 3 7 6 4 4 4 4 6 9 4 9 6 5 1 5 5 8 9 6 2 9 6 4 2 6 7 1 6 8 4 7 3 1

```
[jing.yan@hpc-head-n2 ~]$ cat sudo.sh.o8205131
workID is 32, the number of solutions is 0!
workID is 6, the number of solutions is 0!
workID is 26, the number of solutions is 0!
workID is 16, the number of solutions is 0!
workID is 12, the number of solutions is 0!
workID is 28, the number of solutions is 0!
workID is 8, the number of solutions is 0!
workID is 15, the number of solutions is 0!
workID is 11, the number of solutions is 0!
workID is 29, the number of solutions is 0!
workID is 13, the number of solutions is 0!
workID is 14, the number of solutions is 0!
workID is 31, the number of solutions is 0!
workID is 17, the number of solutions is 0!
workID is 7, the number of solutions is 0!
workID is 18, the number of solutions is 0!
workID is 19, the number of solutions is 0!
workID is 10, the number of solutions is 0!
workID is 30, the number of solutions is 0!
workID is 5, the number of solutions is 0!
workID is 24, the number of solutions is 0!
workID is 25, the number of solutions is 0!
workID is 23, the number of solutions is 0!
workID is 1, the number of solutions is 0!
workID is 20, the number of solutions is 0!
workID is 21, the number of solutions is 0!
workID is 2, the number of solutions is 76764!
workID is 27, the number of solutions is 115400!
workID is 9, the number of solutions is 197132!
workID is 22, the number of solutions is 379888!
workID is 4, the number of solutions is 633912!
workID is 3, the number of solutions is 509052!
The num of processes is 32, the num of solutions is 1912148, total time is 18745 ms.
```