Practice 8. Threads

Variant 1—Sudoku Solution Validator

A Sudoku puzzle uses a 9×9 grid in which each column and row, as well as each of the nine3×3 sub grids, must contain all of the digits 1···9. Following figure presents an example of a valid Sudoku puzzle. This lab consists of designing a multithreaded application that determines whether the solution to a Sudoku puzzle is valid. There are several different ways of multithreading this application. One suggested strategy is to create threads that check the following criteria:

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

I. Passing Parameters to Each Thread

The parent thread will create the worker threads, passing each worker the location that it must check in the Sudoku grid. This step will require passing several parameters to each thread. The easiest approach is to create a data structure using a `struct`.

II. Returning Results to the Parent Thread

Each worker thread is assigned the task of determining the validity of a particular region of the Sudoku puzzle. Once a worker has performed this check, it must pass its results back to the parent. One good way to handle this is to create an array of integer values that is visible to each thread. The i^{th} index in this array corresponds to the i^{th} worker thread. If a worker sets its corresponding value to 1, it is indicating that its region of the Sudoku puzzle is valid. A value of 0 indicates otherwise. When all worker threads have completed, the parent thread checks each entry in the result array to determine if the Sudoku puzzle is valid.

```
sudoku = [
    [6, 2, 5, 8, 4, 3, 7, 9, 1],
    [7, 9, 1, 2, 6, 5, 4, 8, 3],
    [4, 8, 3, 9, 7, 1, 6, 2, 5],
    [8, 1, 4, 5, 9, 7, 2, 3, 6],
    [2, 3, 6, 1, 8, 4, 9, 5, 7],
    [9, 5, 7, 3, 2, 6, 8, 1, 4],
    [5, 6, 9, 4, 3, 2, 1, 7, 8],
    [3, 4, 2, 7, 1, 8, 5, 6, 9],
    [1, 7, 8, 6, 5, 9, 3, 4, 2]
]


# sudoku = [
#     [10, 3, 14, 5, 4, 6, 11, 1, 8, 15, 13, 9, 16, 7, 12, 2],
#     [15, 8, 13, 9, 16, 12, 2, 7, 3, 10, 14, 5, 4, 1, 12, 11],
#     [1, 6, 11, 4, 9, 8, 13, 15, 12, 7, 2, 16, 5, 10, 3, 14],
#     [7, 12, 2, 16, 5, 3, 14, 10, 6, 1, 11, 4, 9, 15, 8, 13],
#     [14, 5, 7, 12, 3, 4, 10, 11, 9, 13, 1, 6, 8, 2, 16, 15],
#     [13, 9, 1, 6, 8, 16, 15, 2, 5, 14, 7, 12, 3, 11, 4, 10],
#     [11, 4, 10, 3, 6, 9, 1, 13, 16, 2, 15, 8, 12, 14, 5, 7],
#     [2, 16, 15, 8, 12, 5, 7, 14, 4, 11, 10, 3, 6, 13, 9, 1],
#     [12, 2, 16, 15, 7, 14, 5, 3, 11, 6, 4, 10, 1, 8, 13, 9],
#     [6, 11, 4, 10, 1, 13, 9, 8, 2, 12, 16, 15, 7, 3, 14, 5],
#     [3, 14, 5, 7, 10, 11, 4, 6, 13, 8, 9, 1, 15, 12, 2, 16],
#     [8, 13, 9, 1, 15, 2, 16, 12, 14, 3, 5, 7, 10, 6, 11, 4],
#     [5, 7, 12, 2, 14, 10, 3, 4, 1, 9, 6, 11, 13, 16, 15, 8],
#     [4, 10, 3, 14, 11, 1, 6, 9, 15, 16, 8, 13, 2, 5, 7, 12],
#     [9, 1, 6, 11, 13, 15, 8, 16, 7, 5, 12, 2, 14, 4, 10, 3],
#     [16, 15, 8, 13, 2, 7, 12, 5, 10, 4, 3, 14, 11, 9, 1, 6]
# ]
```

Here I created dynamically matrix. 9x9 and 16x16 matrix

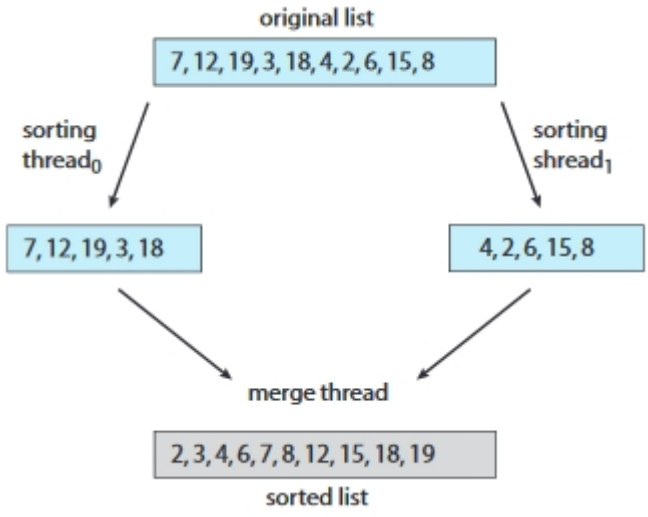
```
In [102]:
with Pool(cpu_count()) as p:
    result = p.map(
        multi_run_wrapper, [(sudoku,i) for i in range(len(sudoku))])

    print(result)
    if all(result):
        print("This is sudoku")
    else:
        print("This is not sudoku")
```

```
[True, True, True, True, True, True, True, True, True]
This is sudoku
```

According to size of matrix I create threads. For example for 16x16 matrix I create 16 threads.

Variant-2. Multithreaded Sorting Application

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term sorting threads) sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread —a merging thread—which merges the two sublists into a single sorted list. Because global data are shared across all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. Graphically, this program is structured as in following figure. This programming project will require passing parameters to each of the sorting threads. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting. Refer to the instructions in Project1 for details on passing parameters to a thread. The parent thread will output the sorted array once all sorting threads have exited.



Multithreaded sorting

1. A thread that divides the list into exactly two parts

2. A thread sort right side

3. A thread sort left side

4. A thread combine two sorted sides

5. A thread sort all list