PDC PROJECT REPORT:

GROUP MEMBERS: 19K-0155,19K-1288,19K-0232

SECTION: G

SUBMITTED TO: Dr. Nadeem Kafi.

INTRODUCTION+OBJECTIVE:

The project that we have worked on is a Sudoku solver, the unique thing in this project is that we have applied a parallel programming paradigm in which we have used OPENMP and MPI programming techniques. A general overview about the project is described as follows, A standard Sudoku contains 81 cells, in a 9×9 grid, and has 9 boxes, each box being the intersection of the first, middle, or last 3 rows, and the first, middle, or last 3 columns. Each cell may contain a number from one to nine, and each number can only occur once in each row, column, and box. A Sudoku starts with some cells containing numbers (*clues*), and the goal is to solve the remaining cells. Proper Sudoku have one solution. Players and investigators use a wide range of computer algorithms to solve Sudoku, study their properties, and make new puzzles, including Sudoku with interesting symmetries and other properties.

CODE EXPLANATION:

A standard Sudoku contains 81 cells, in a 9×9 grid, and has 9 boxes, each box being the intersection of the first, middle, or last 3 rows, and the first, middle, or last 3 columns. Each cell may contain a number from one to nine, and each number can only occur once in each row, column, and box. A Sudoku starts with some cells containing numbers (clues), and the goal is to solve the remaining cells. Given a partially filled 9×9 2D array 'grid[9][9]', the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and sub-grid of size 3×3 contains exactly one instance of the digits from 1 to 9. The naive approach is used to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. We tried every configuration one by one until the correct configuration is found, i.e., for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned position check if the matrix is safe or not. If safe print else recurs for other cases. The unassigned positions are filled with zeros inititally.

Algorithm:

- Create a function that checks if the given matrix is valid sudoku or not. Keep Hashmap for the row, column and boxes. If any number has a frequency greater than 1 in the HashMap return false else return true;
- Create a recursive function that takes a grid and the current row and column index.
- Check some base cases. If the index is at the end of the matrix, i.e. i=N-1 and j=N then check if the grid is safe or not, if safe print the grid and return true else return false. The other base case is when the value of column is N, i.e., j = N, then move to next row, i.e. i++ and j = 0.
- if the current index is not assigned then fill the element from 1 to 9 and recur for all 9 cases with the index of next element, i.e., i, j+1. if the recursive call returns true then break the loop and return true.
- if the current index is assigned then call the recursive function with index of next element, i.e., i, j.

Complexity Analysis:

Time complexity: O(9^(n*n)).

For every unassigned index, there are 9 possible options so the time complexity is O(9^(n*n)).

Space Complexity: O(n*n).

To store the output 2D array a matrix is needed.

METHODOLOGY:

ABOUT OPENMP:

OPENMP is typically used for loop-level parallelism, but it also supports function-level parallelism. This mechanism is called OPENMP sections. The structure of sections is straightforward and can be useful in many instances. Consider one of the most important algorithms in computer science, the quicksort.

How OPENMP Relates to this Project:

Multiple OPENMP techniques were used by us in the project, amongst them were

- Omp for
- Omp atomic
- Omp critical
- Omp nowait

A brief description about each of the following type of OpenMP technique has been penned down below.

OPENMP atomic:

The **OPENMP atomic** directive allows access of a specific memory location atomically. It ensures that race conditions are avoided through direct control of concurrent threads that might read or write to or from the particular memory location. With the **OPENMP atomic** directive, you can write more efficient concurrent algorithms with fewer locks.

Objects that can be updated in parallel and that might be subject to race conditions should be protected with the **omp atomic** directive. All atomic accesses to the storage locations designated by x throughout the program should have a compatible type. Within an atomic region, multiple syntactic occurrences of x must designate the same storage location. All accesses to a certain storage location throughout a concurrent program must be atomic. A non-atomic access to a memory location might break the expected atomic behaviour of all atomic accesses to that storage location. Neither v nor expr can access the storage location that is designated by x. Neither x nor expr can access the storage location that is designated by v.

All accesses to the storage location designated by x are atomic. Evaluations of the expression expr, v, x are not atomic.For atomic capture access, the operation of writing the captured value to the storage location represented by v is not atomic.

OPENMP critical:

where *names* can optionally be used to identify the critical region. Identifiers naming a critical region have external linkage and occupy a namespace distinct from that used by ordinary identifiers. A thread waits at the start of a critical region identified by a given name until no other thread in the program is executing a critical region with that same name. Critical sections not specifically named by omp critical directive invocation are mapped to the same unspecified name.

OPENMP nowait:

If there are multiple independent loops within a parallel region, you can use the nowait clause to avoid the implied barrier at the end of the loop construct.

ABOUT OPENMPI:

MPI is a Message Passing Interface project combining technologies and resources from several other projects. It helps us to communicate with other process within a same network. The MPI standard is created and maintained by the MPI Forum, an open group consisting of parallel computing experts from both industry and academia. MPI defines an API that is used for a specific type of portable, high-performance inter-process communication (IPC): message passing. Specifically, the MPI document describes the reliable transfer of discrete, typed messages between MPI processes. Although the definition of an "MPI process" is subject to interpretation on a given platform, it usually corresponds to the operating system's concept of a process (e.g., a POSIX process). MPI is specifically intended to be implemented as middleware, meaning that upper-level applications call MPI functions to perform message passing. MPI defines a high-level API, meaning that it abstracts away whatever underlying transport is actually used to pass messages between processes. The first version of the MPI standard, MPI-1.0, was published in 1994 [Mes93]. MPI-2.0, a set of additions on top of MPI-1, was completed in 1996. In the first decade after MPI-1 was published, a variety of MPI implementations sprung up. Many were provided by vendors for their proprietary network interconnects. Many other implementations arose from the research and academic communities. Such implementations were typically "research-quality," meaning that their purpose was to investigate various high-performance networking concepts and provide proofs-of-concept of their work. However, some were high enough quality that they gained popularity and a number of users.

How OPENMPI Relates to this Project:

OPENMPI can be related to this project as we are implementing the Sudoku solver 9x9 grid by passing the 9x9 Sudoku array using MPI and its clauses on a single node. We are implementing the parallel Sudoku game and runs on a different test cases. We will use MPI_send() function to send the array from master process to other processes in a COMM_WORLD cluster and other processes receive them using MPI_recv() function and implements different calculation upon it. We will first describe the size of the COMMWORLD that is how many processes run upon the cluster and each process have the rank which tells that who is the master process and who is the slave one. OPENMPI can further be sending data of Sudoku to multiple process and gathers them at single node after applying the logic of the code.

PROBLEMS FACED:

As we know that OPENMPI runs on independent projects so that it can easily be decomposed but in the Sudoku problem we are facing the issue that it is dependent upon its previous rows that when it fills it can further move towards other rows for completion but it cannot be run in a parallel way it has a lot of issue in decomposing the grid among different processes so we are running it on a single node and applies the MPI upon it. However, it may run on a MPI process and compile with it but there is always a problem of solving the Sudoku with MPI. The parallel version takes much longer to run than the sequential, because the sequential version stops when you find a solution, and the parallel version traverses the whole search space. And breaking out of a parallel search is hard. It's the same problem that you cannot parallelize a while loop with OPENMP.

Comparison via Graphs:

We ran both programs and noted the time values from start till end and noted clock tick of intermediate program of open mpi sudoku solver and write all values to a file named ‘open_mp_time.txt’ and we did same for the mpi program for sudoku solver and write all values to a file named ‘mpi_time.txt’ and then plot a graph with the help of matplotlib. Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. As this is a python-based library so we installed python in our VM of Ubuntu 20.04 LTS and run it on pycharm

compiler or Google Collab. After running it on Pycharm or Google Collab we can observe the graphical representation of time and count. We plot time on y-axis as it is independent and count of program on x-axis. Finally we observed curves and based on that curves we conclude that which one is faster either Openmp or mpi in terms of time.

CONCLUSION:

The conclusion is that the project we are doing is the parallel Sudoku solver in which we are suing OPENMP AND OPENMPI which we study in our PDC course for parallelization of multiple processes so that we can decrease the time complexity for solving the problems and we are completing our project in the motivation to build the Sudoku in a parallel format.