National University
of computer and emerging sciences

PROJECT PROPOSAL
Parallel & Distributed Computing

Project Name:

Parallel Sudoku Solver

Section: J

Team Members:

19K - 1331 AliHasnain Ladhani (Lead)

19K - 1396 Abdul Samad

19K – 0330 Ziyaan Ali

Advisor:

Miss Nausheen Shoaib

OBJECTIVE:

Using OpenMP for Parallel and Distributed Computing course. For this purpose, we will be developing a serial and parallel implementation of a sudoku solver.

METHODOLOGY:

The puzzle consists of a matrix of size n x n which is partially filled and the algorithm fills the matrix cells which are blank with values from 1 to n in such a way that no value is repeated more than once on each of n columns and n rows or n squares of size $\sqrt{n} \times \sqrt{n}$ on which the matrix is split.

The algorithm we are using can be summarized as follows:

1. **Apply heuristics** on the input grid. This gives the initial grid for creation of allotment list.
2. **Create a list of grids for allotment** among the threads by doing BFS level-by-level on a tree of intermediate grids that has the initial grid at the root. This is done till there are less than thread count many grids in the list.
3. **Each thread** gets a grid from the allotment list and uses a **local stack** to **execute brute force DFS** on it. It then **repeats** the following till either the solution is found or the stack is empty in which case it gets the next grid.
    a. Pop a grid from the search stack.
    b. Apply heuristics on it.
    c. Expand the tree by selecting the cell with least number and pushing the newly created grids into the stack.
    d. Whenever the solution is found the thread sets **a shared variable** indicating this and all the threads exit the parallel section.
4. After the parallel section, the value of the shared variable is checked to figure out if the solution was found.

The design decisions that we took in our implementations include:

1. **Possible values** for each cell in the grid store as a **bit mask of length 64**.
    - **O(1) addition, deletion and searching etc.** of possible values and hence is much faster than using an array.
    - Other operations such as **getting number of possible values etc.** also done in O(1) time using gcc's builtin functions.
2. The application of heuristics is done as follows till none of the heuristics make any change:
    a. Sequence in which the heuristics are applied: **Elimination -> Loneranger -> Twins**.
    b. If the application of a heuristics causes some change, then the sequence is repeated from the beginning. This ensures that elimination and lone rangers being more useful are applied more frequently.
3. **Static allocation** of workload among the threads as dynamic allocation didn't seem to be useful.

a. Grids assigned in **round-robin fashion** for similar workloads on all threads.

4. **Prune the DFS** tree i.e., ignore the branch whenever:
    a. A cell has no possible values
    b. A number doesn't occur in the possible values of any cell in a row/column/box.
5. Stacks store any grid that gets freed so that a new grid doesn't need to be allocated from scratch.
6. **Parallelized only the DFS section**, as each application of the heuristics doesn't take much time and so the overheads of parallelizing them would have been too much.
7. Triplets not used as it isn't efficient enough.