Parallel Programming – MPI and OpenCL Sudoku Solver

PROJECT SYNOPSIS

Ninad S Shetty (Sec 'A' - 140905103) Prajwal P (Sec 'A' - 140905123)

6th Semester, CSE, Manipal Institute of Technology

28th FEBRUARY, 2017

INTRODUCTION

Sudoku (originally called **Number Place**), is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub grids that compose the grid (also called "boxes", "blocks", "regions", or "sub squares") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a unique solution.

The general pseudo code of backtracking algorithm for standard Sudoku template (9x9.)

```
Initialize 2D array with 81 empty grids (nx = 9, ny = 9)
Fill in some empty grid with the known values
Make an original copy of the array
Start from top left grid (nx = 0, ny = 0), check if grid is empty
if (grid is empty) {
  assign the empty grid with values (i)
  if (no numbers exists in same rows & same columns same as (i) & 3x3 square (i) is
currently in)
    fill in the number
  if (numbers exists in same rows | same columns same as (i) | 3x3 square (i) is currently
in)
    discard (i) and repick other values (i++)
}
else {
  while (nx < 9) {
    Proceed to next row grid (nx++, ny)
    if (nx equals 9) {
      reset nx = 1
      proceed to next column grid(nx,ny++)
```

```
    if (ny equals 9) {
      print solution
  }}}}
```

ABSTRACT

Total number of valid Sudoku puzzles is approximately 6.671×1021 (9*9 matrix).
Trying to populate all these grids is itself a difficult problem because of the huge number of possibilities and combinations of puzzles, Assuming each solution takes 1 micro second to be found, then with a simple calculation we can determine that it takes **211,532,970,320.3 years** to find all possible solutions. If that number was small, say 1000, it would be very easy to write a Sudoku solver application that can solve a given puzzle in short amount of time. The program would simply enumerate all the 1000 puzzles and compares the given puzzle with the enumerated ones and we don't have to design a parallel algorithm for finding Sudoku solutions. Unfortunately, this is not the case since the actual number of possible valid grids is extremely large so it's not possible to enumerate all the possible solutions. This large number also directly eliminates the possibility of solving the puzzle with brute force technique in a reasonable amount of time.

Therefore, a method for solving the puzzle quickly will be derived that takes advantage of many core architecture on modern and future computer systems which will bring down the time cost of solving an *NxN* Sudoku to a reasonable amount.

Our goal is to allow for the algorithm to take advantage of the many-core architecture to further reduce its running time. This is very important and is at the heart of our work because for large *N*, the number of cells per row and column in the Sudoku grid, finding a solution becomes an extremely difficult and computationally intensive problem. Hence, the parallel algorithm must also scale as the grid size increases.

ABSTRACT OF PROPOSED ALGORTIHM