What is Sudoku!

- Logic puzzle
- Given a grid: player can deduce all the remaining symbols

Rules: Must have 9 unique symbols (1 to 9)
1. Each Row
2. Each Column
3. Each 3x3 Block
4. All Numbers from 1 – 9
5. Fill in all the blank spots

Standard Sudoku Grid: 9 X 9
Solving a Sudoku

• Two Recursive Steps

1. Constraint Propagation
   • Reduce the amount of possibilities for each cell to 1 number!

2. Search
   • A cell is chosen to assume one of its possible values, then Constraint Propagation is repeated.
Constraint Propagation

• Rule 1
  a. For any cell, if a number already exists in its row, column or box (the cell’s peers), the possibility of that number for that cell is removed.
Constraint Propagation

• Rule 2
  a. For any cell, if all of its peers has a specific number removed, the cell itself must contain that number.
Search

• A Single cell is chosen to assume one of its possible values.
• Constraint_prop()

• If (assumption is TRUE) \(\rightarrow\) eventually arrive at the solution.

• If (assumption is FALSE) or we reach a contradiction \(\rightarrow\) Initial assumption was wrong.

• Remove that assumption from the possibilities list.
Recursive Calls

- CP() -> Search() -> CP() -> Search() ...
Parallel Solution

- Parallelizing Constraint Propagation
Approach

• 1 Master + n worker nodes
• Master inputs a number based on constraints.
• Distributes the grid amongst the workers.
• Workers perform constraint_propagation()
• Masters gathers all the data.
• Repeat till all entries have been made.
Important:

- Please note – chosen inter process communication over efficiency

- Dell – 2.40 Gz Intel Xeon E5645 (Batch System)
  - 372 Total Nodes
  - 12 Cores each
  - Main Memory : 48 GB

- 1 Core / Node

- Why?
  - MPI handles send recv automatically.
  - Cores on the same node use quickest communication medium = shared memory.
  - For Uniformity.
Experiment 1

• Keeping Data Constant and Increasing the Number of Nodes (Processors)
• Data = 50 Easy Sudoku + 90 Hard Sudoku
• Easy = Given ~ (25 to 30)
• Difficult = Given ~ (19 to 25)
• 4 Rounds
  1. Serial
  2. 3x3 cell - > Each Node
50 Easy - 9 x 9 Sudoku

<table>
<thead>
<tr>
<th>Number of Processors (1 Core / Node)</th>
<th>Series3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Node</td>
<td>0.5</td>
</tr>
<tr>
<td>9 Nodes (1 core each)</td>
<td>0.39</td>
</tr>
<tr>
<td>27 Nodes (1 core each)</td>
<td>0.8</td>
</tr>
<tr>
<td>81 Nodes (1 core each)</td>
<td>1.15</td>
</tr>
</tbody>
</table>
90 Hard – 9 x 9 Sudoku

<table>
<thead>
<tr>
<th>Number of Processors (1 Core / Node)</th>
<th>1 Node</th>
<th>9 Nodes (1 core each)</th>
<th>27 Nodes (1 core each)</th>
<th>81 Nodes (1 core each)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Series3</td>
<td>3.8</td>
<td>2.622</td>
<td>4.56</td>
</tr>
</tbody>
</table>

**Execution Time (s)**

**Graph**
- Node
- 9 Nodes (1 core each)
- 27 Nodes (1 core each)
- 81 Nodes (1 core each)
Experiment 2 – Speed Up

• Initially Idea – Run many 25 x 25 Sudoku boards
• Problems with 25 x 25 – Take too long!
• A 9 x 9 is solved really fast
• Best size for analysis – 16 x 16 hard
• Hard -> 104 – 115 cells are filled (16 * 16 = 256)
Execution Time – 16 x 16 Hard Sudoku Board

<table>
<thead>
<tr>
<th>Number of Cores (1 Core / Node)</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Node</td>
<td>74.6749</td>
</tr>
<tr>
<td>2 Nodes</td>
<td>37.5634</td>
</tr>
<tr>
<td>4 Nodes</td>
<td>16.2995</td>
</tr>
<tr>
<td>8 Nodes</td>
<td>7.4974</td>
</tr>
<tr>
<td>16 Nodes</td>
<td>9.6228</td>
</tr>
</tbody>
</table>
Speedup – 16 x 16 Hard Sudoku Board

<table>
<thead>
<tr>
<th>Number of Cores (1 Core/Node)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Node</td>
<td>1</td>
</tr>
<tr>
<td>2 Nodes</td>
<td>1.97</td>
</tr>
<tr>
<td>4 Nodes</td>
<td>4.54</td>
</tr>
<tr>
<td>8 Nodes</td>
<td>9.87</td>
</tr>
<tr>
<td>16 Nodes</td>
<td>7.69</td>
</tr>
</tbody>
</table>

- Avg. (Speedup)
- 2 per. Mov.
### Efficiency

<table>
<thead>
<tr>
<th>Number of Nodes (1 Core / Node)</th>
<th>1 Node</th>
<th>2 Nodes</th>
<th>4 Nodes</th>
<th>8 Nodes</th>
<th>16 Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td>1</td>
<td>0.985</td>
<td>1.135</td>
<td>1.2375</td>
<td>0.480625</td>
</tr>
</tbody>
</table>

The graph above shows the efficiency of different numbers of nodes, with each node having 1 core. As the number of nodes increases, the efficiency initially improves but then starts to decline, indicating diminishing returns after a certain point.
Results & Observations:

• Super Linear Speedup
  • Usually Linear Speed up
  • Generally Noticeable in Open MPI – Cache Effect
  • Occurred Due to my implementation – Broadcasting cell values after constraint propagation.

\[ S_p = \frac{T_1}{T_p} \]

• Efficiency > 1 ?
  • Due to Super Linear Speedup

\[ E_p = \frac{S_p}{p} \]

• Balance of Processors used and Data Distribution -> Best Efficiency

• Easy problems are solved too quickly (serially) -> Inaccurate Speedup
  • Difficult to analyze.
Results & Observations:

• Modified Brute Force approach
  • Good Speedup
  • Poor Execution Time
  • Hard Problems: ~7.5 s
  • Expert Problems: exceeded 15 min quota
  • Other implementation took over 6 hours.

• Parallel Programming is really hard! – Very Interesting at the same Time!
References

• Parallelization of Sudoku – (University of Toronto)
  http://individual.utoronto.ca/rafatrashid/Projects/2012/SudokuReport.pdf

• Parallel Sudoku Solver – Carnegie Mellon University, Hilda Huang, Lindsay Zhong

• Arbitrary Size Parallel Sudoku Creation – William Dudziak
  http://www.dudziak.com/ArbitrarySizeSudokuCreation.pdf

• Solving Every Sudoku Puzzle – Peter Norvig
  http://norvig.com/sudoku.html