COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

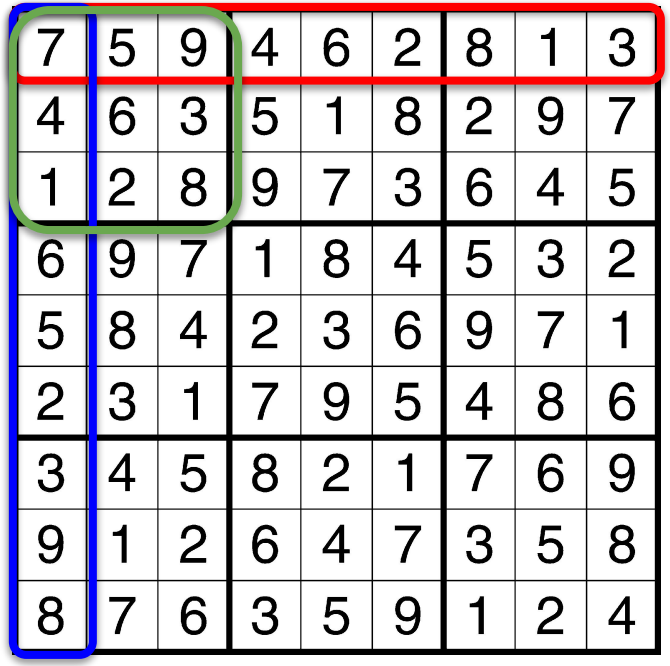
Parallel Sudoku Solver

Haoran Zhou, Yi Liu

Contents

Introduction

Each row, each column, and each 3 X 3 subgrid contains all digits 1 - 9.

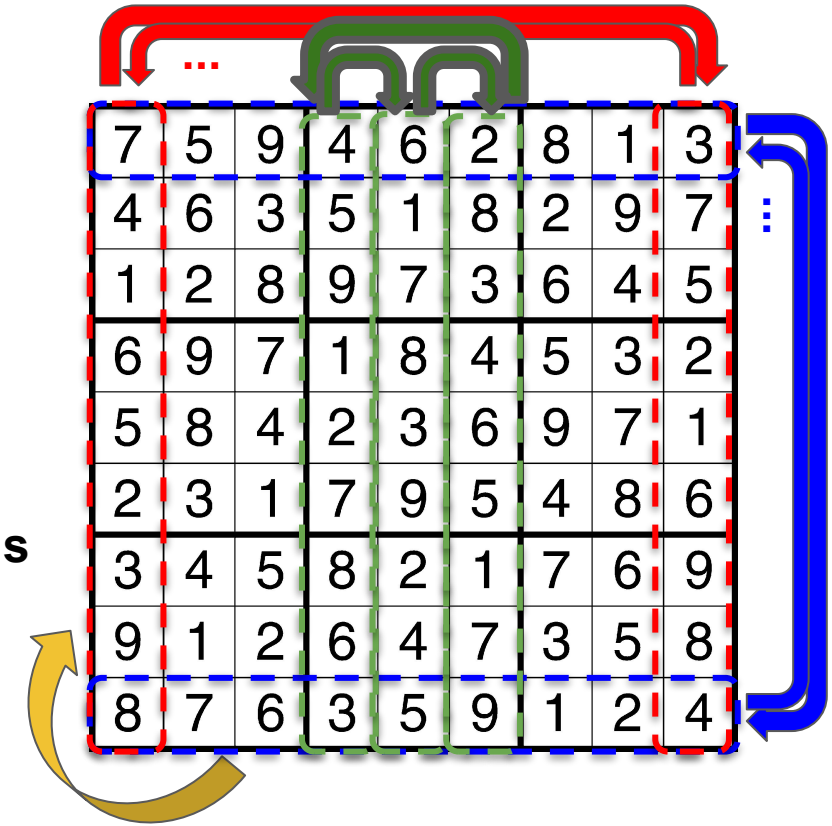


7	5	9	4	6	2	8	1	3
4	6	3	5	1	8	2	9	7
1	2	8	9	7	3	6	4	5
6	9	7	1	8	4	5	3	2
5	8	4	2	3	6	9	7	1
2	3	1	7	9	5	4	8	6
3	4	5	8	2	1	7	6	9
9	1	2	6	4	7	3	5	8
8	7	6	3	5	9	1	2	4

Sudoku Generator

Steps:

1. **Copy a Sudoku Array from online resource**
2. **Flip / rotate for creating new random Sudoku Array**
3. **Random remove defined number of cells in the Sudoku Array**



Sudoku Generator

Steps:

3. Random remove defined number of cells in the Sudoku Array

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

RmNum = 35

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

Sudoku Solver - Serial

We use a stack to tack the DFS tree

- Explore the solution space using depth first search

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

Step 0

Initial problem

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

Step 1

Try 1 at cell (8, 6)

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

Step 2

Try 1 at cell (8, 7)

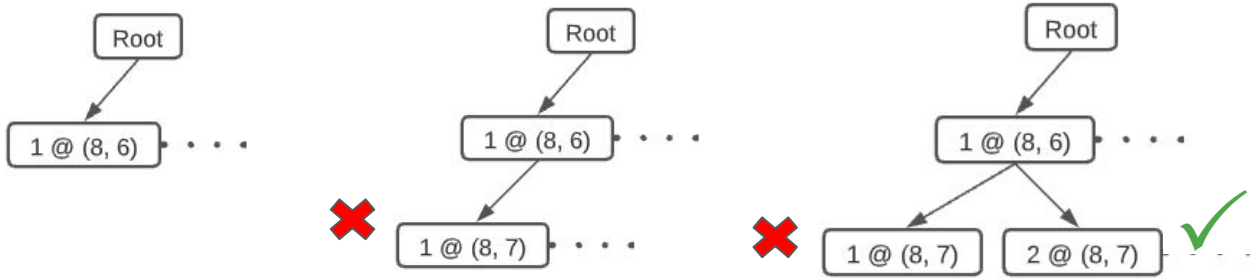
Fail and backtrace

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

Step 3

Try 2 at cell (8, 7)

Success

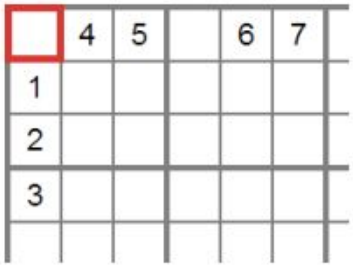


Sudoku Solver - Serial

- **Pruning**

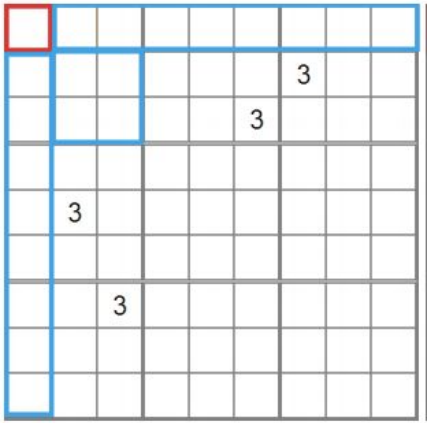
Rule 1

Some possibilities are ruled out by its peers



Rule 2

If all of its peers has a specific number ruled out, the cell itself must contain that number



Sudoku Solver - Serial

Original parallel reference needs several seconds to solve a sudoku problem (TOO SLOW!!!)

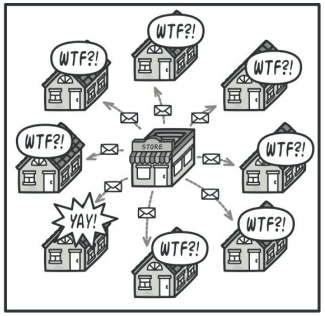
Optimization - Memory Pool

- **Heap allocation: 1000 new operations take around 1 ms, and can cause heap fragmentation**
- **STL containers use new operation by default**
- **Preallocating all the memory makes things faster**

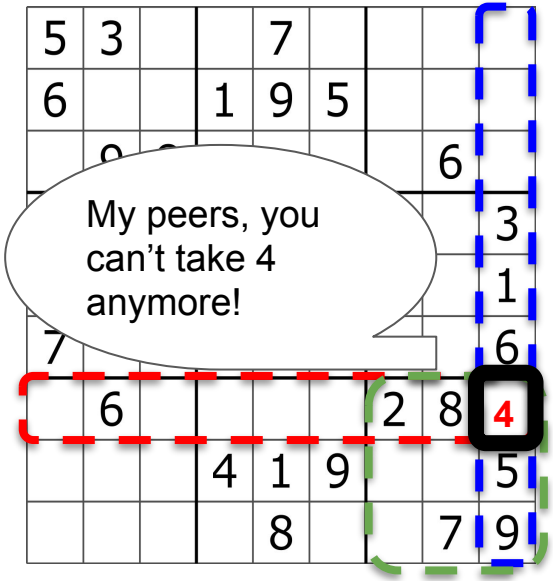
```
MemPool <T> {
    T* Apply();
    void Delete(T*);
}
```

Sudoku Solver - Serial

Optimization - Observer Design Pattern



Prune Rule Propagation



CellState Class

Both subscriber and publisher

```
CellState {
    Subscribe(CellState *);
    Unsubscribe(CellState *);
    NotifyConstraints();
}
```

Elements with values filled should unsubscribe all it's publishers
Faster to broadcast a message!

Modified Subscriber List

O(1) Insertion

O(1) Deletion

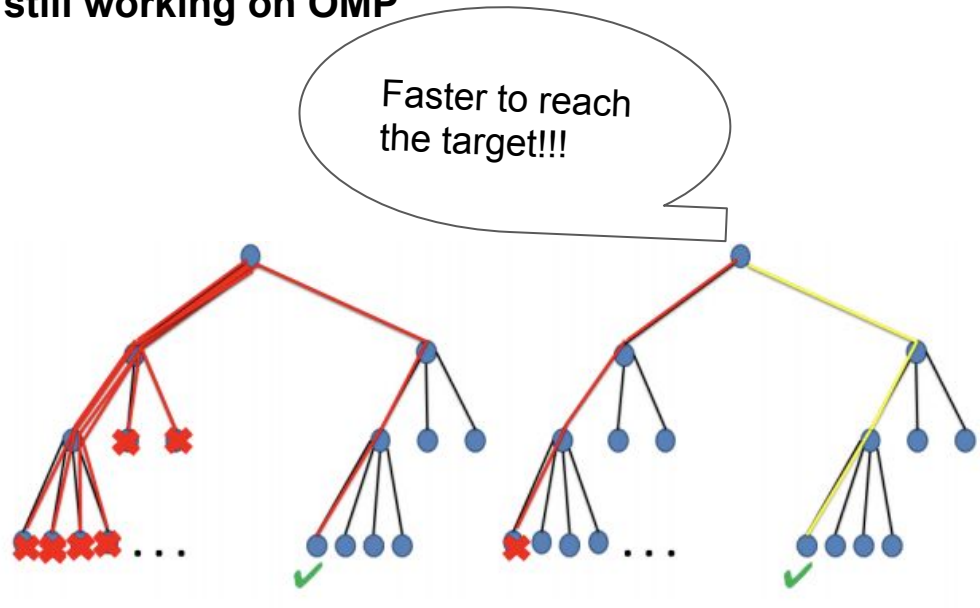
O(1) Lookup

Serial solver speed:
Simple sudoku: less than 1 ms
Most can be solved within 5 ms

Sudoku Solver - Parallel

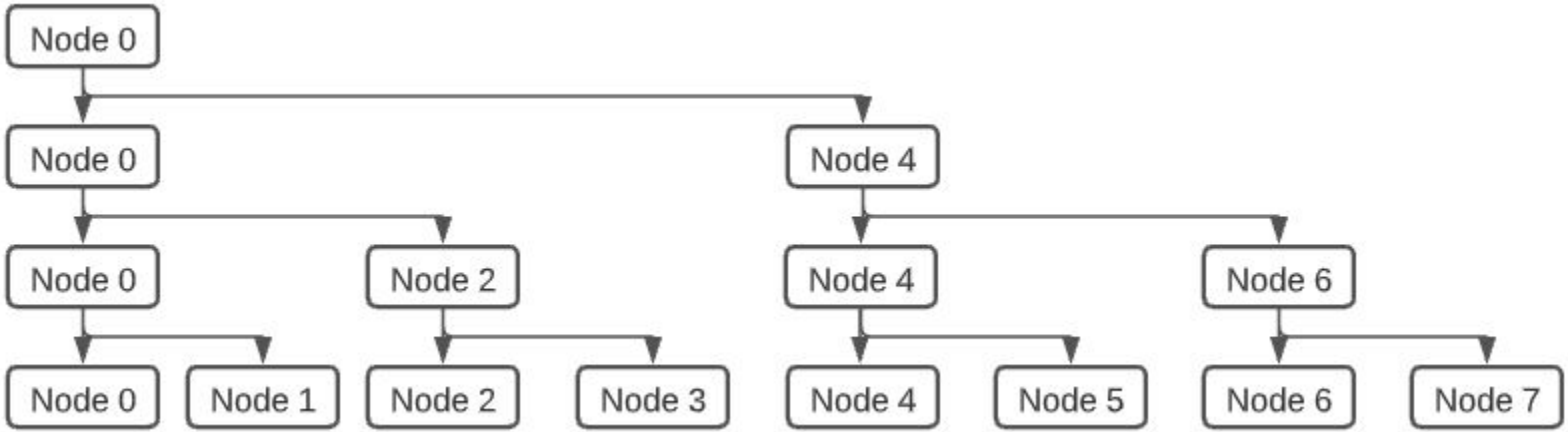
Parallel searching

Finished MPI, still working on OMP



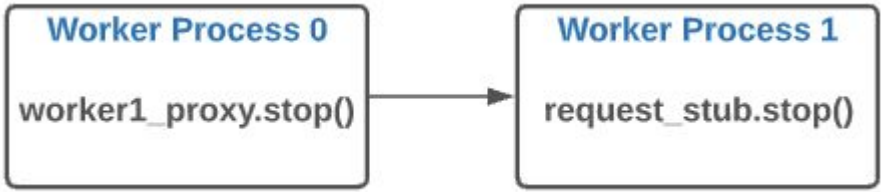
Sudoku Solver - Parallel

Problem distribution O(log(N))



Sudoku Solver - Parallel

A remote procedure call (RPC) style design like Charm++

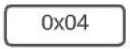



```
WorkerProxy {
    SetProblem(Solvable*);
    PushProblem(Solvable*);
    Stop();
    AskForWork();
    Kill();
    SetConstraints(size_t x_idx, size_t y_idx, int val);
}
```

Example of messages

PUSH_PROBLEM

KILL

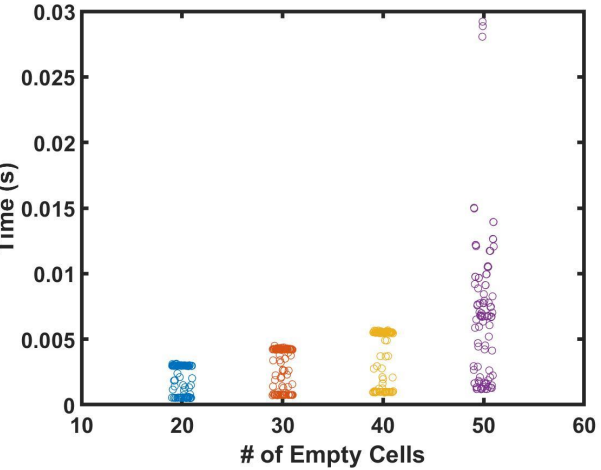


Performance Tests

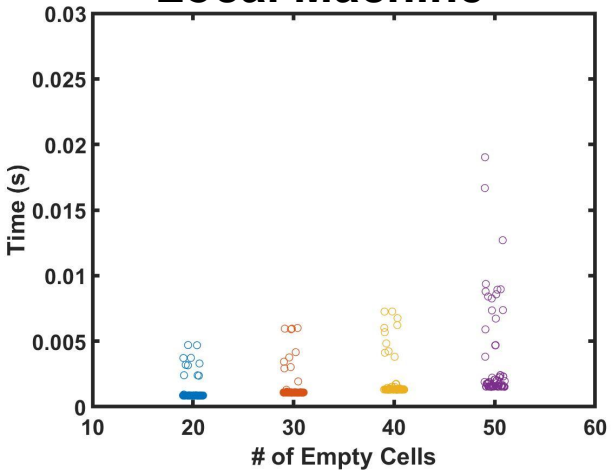
- The following performance tests are conducted on both **local machine** and **Deepthought2**
- Each set-up was experimented with 100 random Sudoku problems
  - # of Process: 1, 2, 4, 8
  - # of Empty Cells: 20, 30, 40, 50

Performance Results - # of Empty Cells

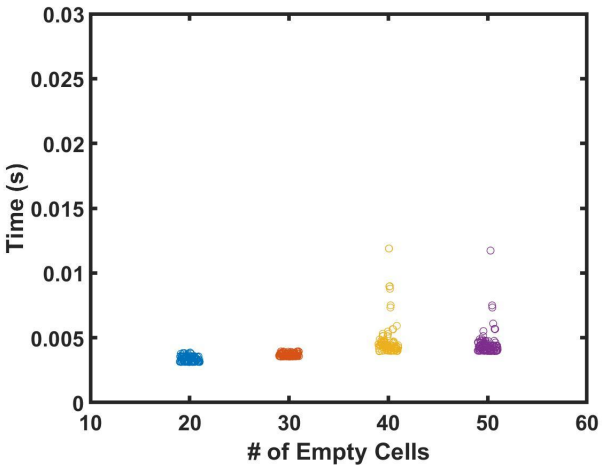
Serial Solver



4 Process on Local Machine



4 Process on Deepthought2



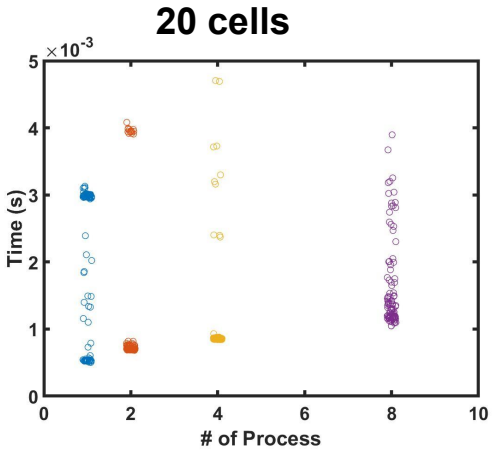
- More empty cells to solve lead to more solving time
- Lower performance variance with MPI parallel solver

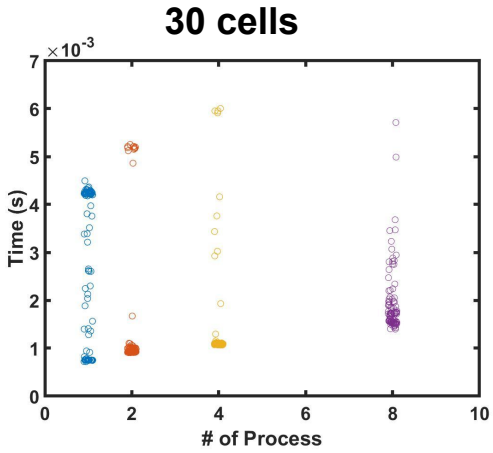
Performance Results - # of Process

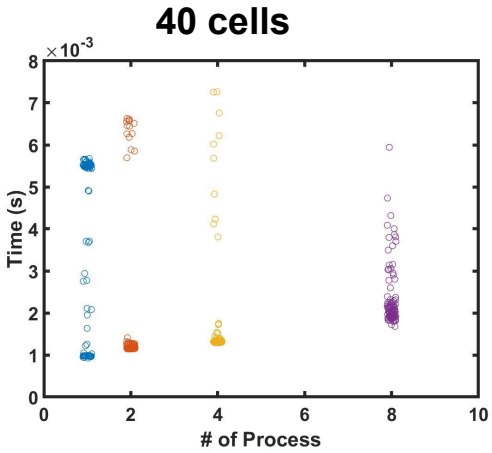
- MPI solver performance is more stable

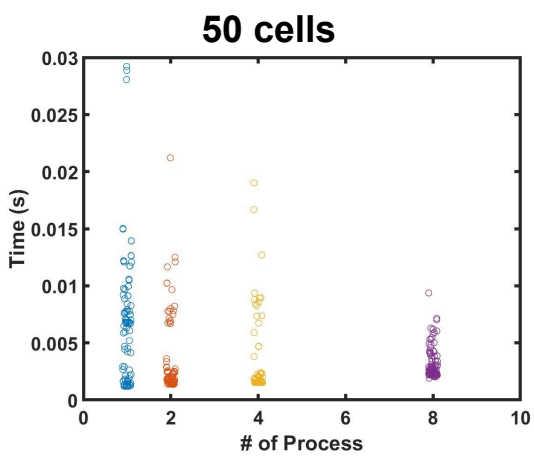
In most cases, MPI model:

- worse than the best performance of serial solver
- But better than bad performance of serial solver





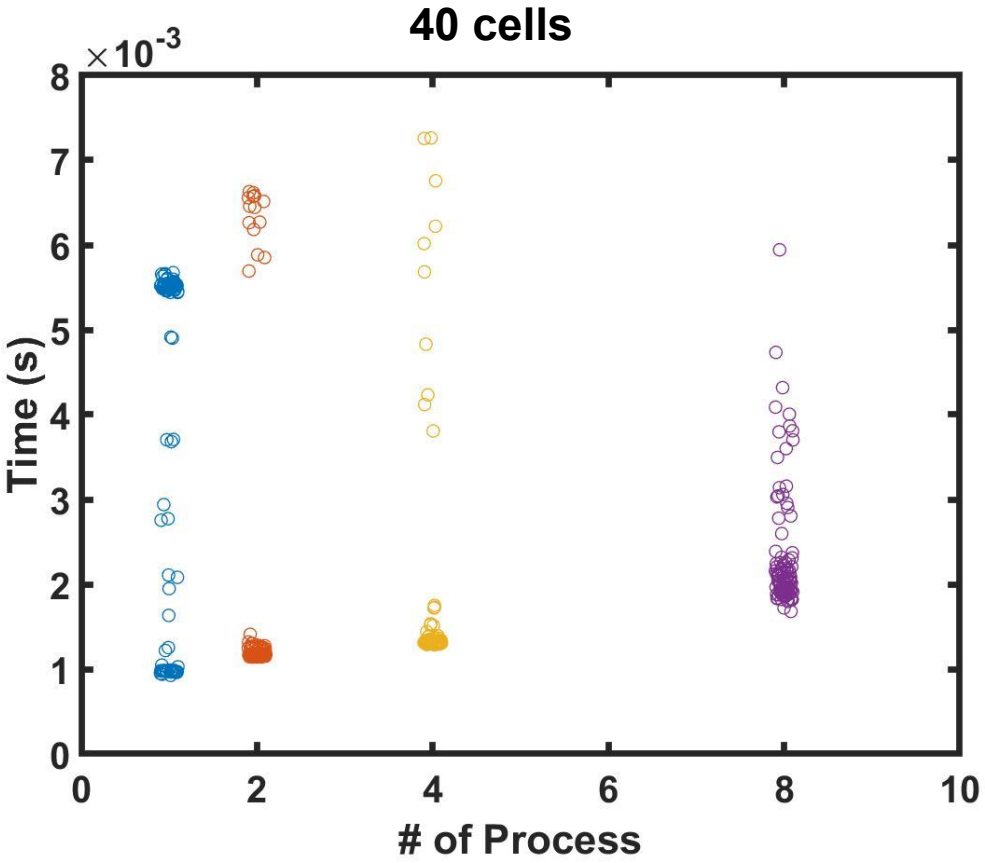




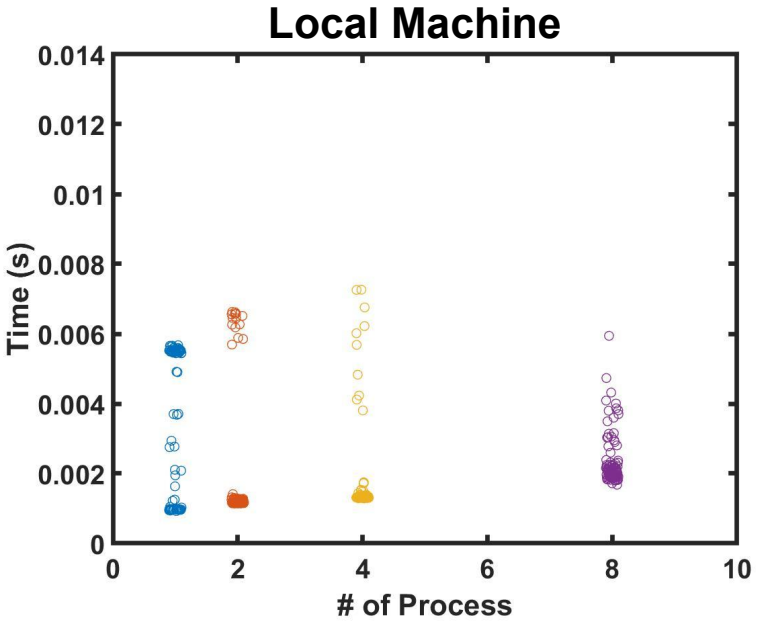
Performance Results - # of Process

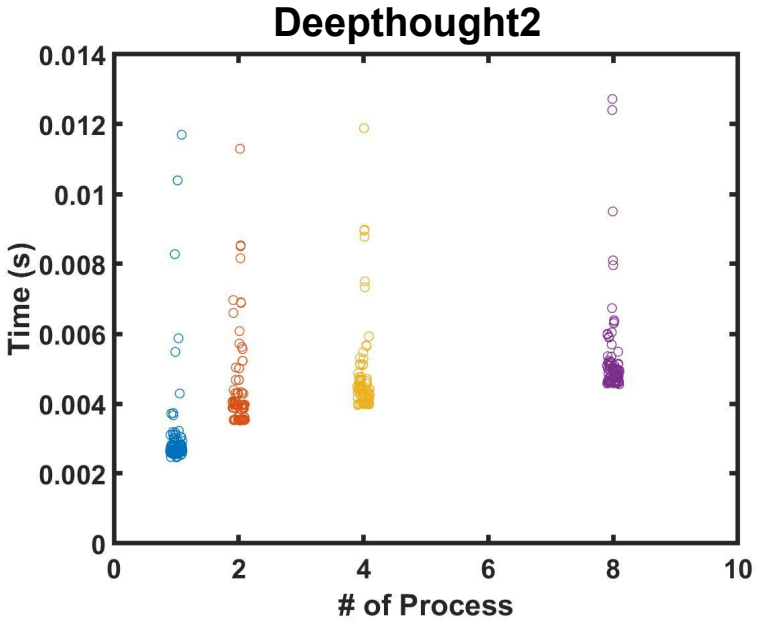
Why?

- **Difficulty of Sudoku problems has huge variance**
- **Some problems are easy to solver by filling the cell with only one possible value one by one**
- **Parallel solver only works well for problem with lots of possible values**



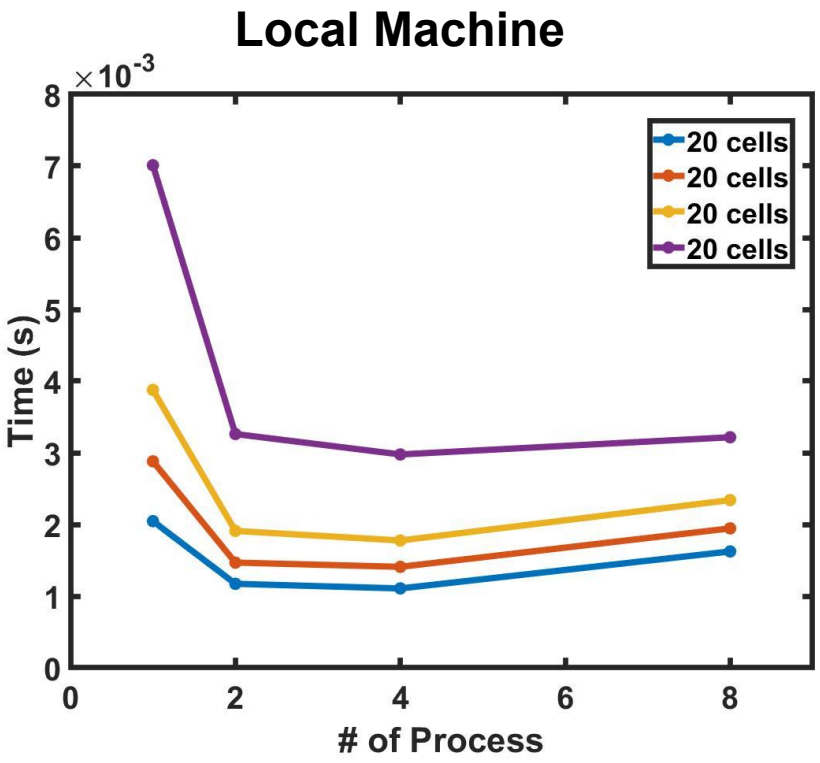
Performance Results - Local Machine vs. Deepthought 2

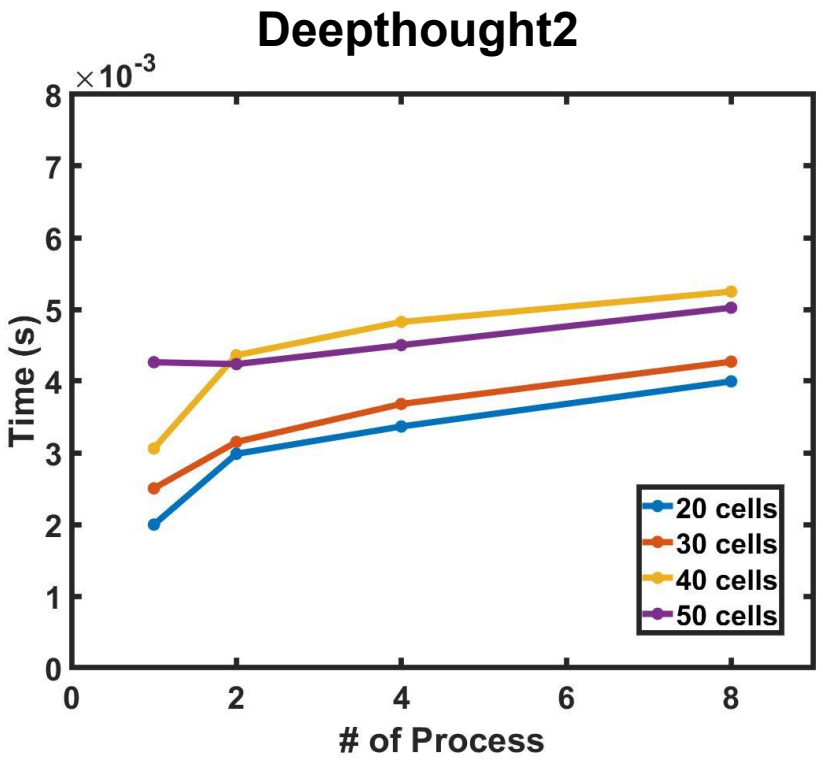




- Local machine has better performance (averaged)
- Deepthought2 has lower performance variance

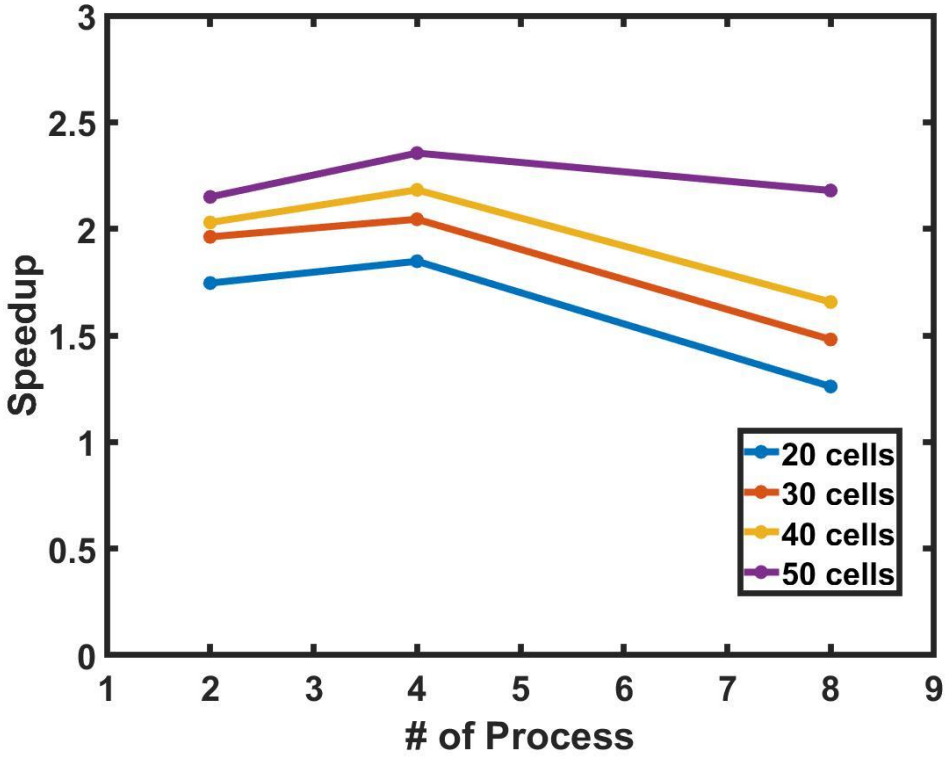
Performance Results - Scaling Performance





Performance Results - Speedup

- 4-process model works the best
- The more cells in the problem to solve, parallel solver works the better



Conclusions

- **Serial solver works better on some easy Sudoku problems**
- **Even though more processes may handle trial tasks simultaneously, the communication takes a lot of time**
- **Communication on Deepthough2 costs more time than on our local machine**
    - **We ran a simple MPI_Send and receive test, local machine took 4e-6 s while dp2 took 7e-5 s**

Question Time