

HPC Report

by Steffan Benson

FILE	1348_STEFFAN_BENSON_HPC_REPORT_107698_756983258.DOCX (146.17K)		
TIME SUBMITTED	08-MAY-2019 03:51PM (UTC+0100)	WORD COUNT	1370
SUBMISSION ID	106668657	CHARACTER COUNT	7362

Assignment Title: Parallelization Report
Module Title: High Performance Computing for Games and Visualisation
Student Name: Steffan John Benson
Student Number: P151818
Lecturer Name: Gordon Dickers & Andrew Baker
Submission Deadline: 08/05/2019

ABSTRACT

The purpose of this project is to test implementations of an algorithm in both linear and parallelized forms to determine the difference in completion time. The algorithm in question is a backtracking method for solving Sudoku puzzles, a grid-based logic puzzle.

Although not specifically used in Sudoku puzzles, backtracking is one of the most common methods used to solve them, both in practical use and through computer programs. Backtracking in computerized solvers are separated into 2 categories, linear and parallelized. Theoretically, the linear algorithm should be more time consuming to execute compared to the parallelized algorithm, as the parallelized variant is able to solve many different grids simultaneously as opposed to the 'one at a time' approach the linear algorithm is restricted to.

Functional implementations of the algorithms are highly complex and are not readily available online, and thus no testing can be performed. Based on the accounts of other developers, linear backtracking is significantly slower than parallelized backtracking, with the parallelized variant at least halving the time taken to solve a puzzle.

TABLE OF CONTENTS

ABSTRACT	I
TABLE OF CONTENTS	II
TABLE OF FIGURES	II
1 INTRODUCTION	3
2 RESEARCH AND REVIEW OF LITERATURE	4
2.1 SUDOKU	4
2.1.1 Background information	4
2.1.2 Rules of Sudoku	4
3 METHODS OF SOLVING SUDOKU PUZZLES	6
3.1 LINEAR BACKTRACKING	6
3.2 PARALLELIZATION OF BACKTRACKING	7
3.2.1 Method	7
3.2.2 Generation kernel	7
3.2.3 Backtracking Kernel	7
3.2.4 Theoretical Speed increase	8
3.2.5 Realistic Speed Increase	8
4 REFERENCES	9

TABLE OF FIGURES

FIGURE 1 - UNSOLVED SUDOKU PUZZLE WITH 30 CLUES	5
FIGURE 2 - SOLVED SUDOKU PUZZLE WITH USER VALUES HIGHLIGHTED IN RED	5
FIGURE 3 - EXAMPLE OF A TREE OF POSSIBILITIES	6

1 INTRODUCTION

As people, we come across puzzles almost constantly in our lives. A measure of a person's intelligence is often quantified by a person's ability to solve puzzles of varying complexity as quick as possible, so much so that recreational puzzle solving has become a common pastime for many. A popular type of puzzle used found frequently in recent years has been the Sudoku, a grid-based logic puzzle. Puzzles of varying complexities present seemingly countless possibilities, providing unique challenges for the most daring puzzle solvers. A variety of different methods have been developed since the puzzle's inception, many of which are applicable both as a manual method or as a program. This project will explore current literature and the methods in which sudoku puzzles are solved, so that a linear and parallelised system can be implemented and compared to one another to determine if there is any difference in the rate in which a puzzle can be solved.

2 RESEARCH AND REVIEW OF LITERATURE

2.1 Sudoku [1]

2.1.1 Background information

The Sudoku puzzle originates from Switzerland, where Leonhard Euler created the “Carre Latin”, a simpler form of the puzzle, in the 18th century AD. Sudoku as we know it today was first published in 1979 by American architect Howard Garns; however, it was popularized by the Japanese in 1986. It was this Japanese popularisation that garnered the name Sudoku, which is an abbreviation of Su-ji wa dokushin ni kagiru, which translates to “the numbers must be single”. The puzzle offers an astounding 6,670,903,752,021,072,936,960 possible combinations of puzzles.

2.1.2 Rules of Sudoku

Sudoku puzzles are made up of 81 cells separated into a 9 by 9 grid, with some cells filled with clue values which cannot be altered. Players are tasked with filling the cells with numbers of value 1 to 9, such that each row, column and 3 by 3 region of the grid features each number within that range just once.

Clues are distributed randomly and must conform to the rules of the puzzle. A maximum of 40 clues may be included in a puzzle and a minimum of 17 clues, with any more or less than these values respectively will either make the puzzle too complex or too simple to solve.

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

Figure 1 - Unsolved Sudoku puzzle with 30 clues

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

Figure 2 - Solved Sudoku puzzle with user values highlighted in red

3 METHODS OF SOLVING SUDOKU PUZZLES

3.1 Linear Backtracking[2][3]

Backtracking is an algorithm for determining solutions to a problem by incrementally building and testing candidate solutions. Once a candidate solution has been built, it is tested for validity; and should it fail, the candidate is rejected. Once a candidate is rejected, the test will return to the last positive test candidate, which gives the algorithm its name.

Backtracking can be represented by a tree of possibilities utilising a depth first search method. The root node of the tree is the first empty cell, for example an empty cell in the top left corner of the grid. The first branch in the tree will be inputting the value 1 into the cell. Should the test return positive, a branch from the leaf node will perform the same test. Should the test return negative, the next leaf node will be tested along the current branch, for example replacing the value 1 with a 2.

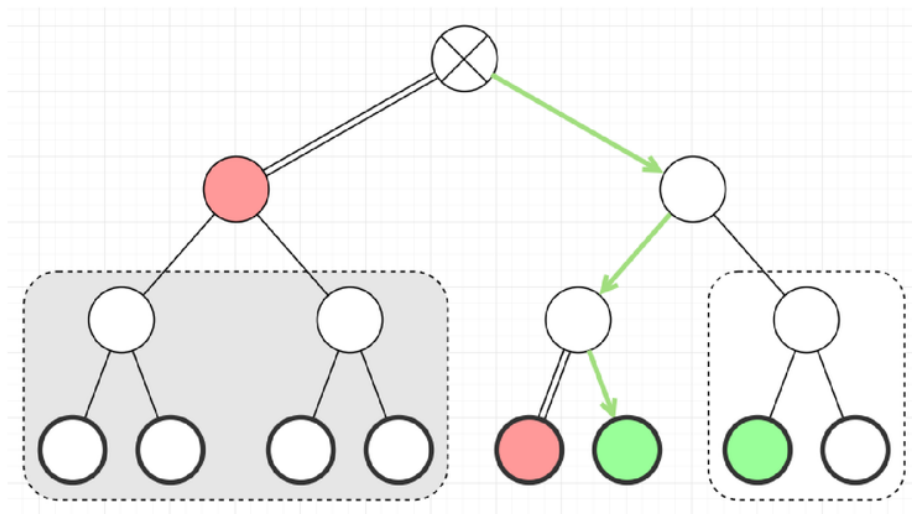


Figure 3 - Example of a tree of possibilities

In figure 3, white circles represent possible candidates, green circles represent successful candidates and red circles signify rejected candidates. The single lines connecting circles signify potential paths for the solution, doubled lines signify paths that have been backtracked due to rejected solutions and green arrows signify successful pathways for the solution.

3.2 Parallelization of Backtracking[4]

The software being discussed here is available on Github.com, however this implementation will not work on any programming environment available to the user at UWTSD.

3.2.1 Method

The method employed by Duang and Teng in their Parallelization of Backtracking program can be broken down into 2 key stages. Stage 1 performs a breadth first search across the unsolved grid to determine all possible grids with a specified number of empty cells filled. Stage 2 will attempt to solve each of these grids separately in threads on the GPU. Once a solution to the grid is determined, the process is terminated, and the solution is returned.

3.2.2 Generation kernel

The program utilises a kernel specifically to generate each separate grid to be searched as well as where these grids are stored. The grid will test each cell for valid inputs. Each valid input will then have a board generated for that possible combination. For example, if a cell has the values 1, 2, 4 & 7 valid for it, the kernel will generate 4 boards to be tested. This operation typically produces around 50 thousand possible boards to be tested.

3.2.3 Backtracking Kernel

The backtracking kernel is responsible for taking the grids generated by the Generation kernel and counting the exact number of grids to be searched. For each grid, the number and location of empty spaces are also determined. Each grid is given a thread which will perform linear backtracking as described above. Once a solution is determined, a global flag notifies all other threads of the solution, closing each unsuccessful thread.

3.2.4 Theoretical Speed increase

In theory, as each thread is performing the linear backtracking method individually, the factor by which the solution speed increases is determined by the number of threads being run. This means that the number of grids being tested equals the factor by which the speed increases. As GPUs can handle thousands of threads, this factor should be in the order of thousands.

3.2.5 Realistic Speed Increase

According to Duang and Teng, the parallelized sudoku solver implementation saw an approximate increase in solve speed by a factor of 4 when compared to the linear equivalent. Linear backtracking provided solve times ranging from ~4ms to ~3s and parallelized backtracking saw solve times between ~1ms and ~800ms.

4 REFERENCES

- [1] "What is Sudoku?," 2008. [Online]. Available: <https://www.learn-sudoku.com/what-is-sudoku.html>. [Accessed: 12-Dec-2018].
- [2] AayushChaturvedi, "Backtracking | Introduction - GeeksforGeeks." [Online]. Available: <https://www.geeksforgeeks.org/backtracking-introduction/>. [Accessed: 12-Dec-2018].
- [3] S. Zibbu, "Sudoku and Backtracking – Hacker Noon," *Hackernoon*, 2018. [Online]. Available: <https://hackernoon.com/sudoku-and-backtracking-6613d33229af>. [Accessed: 08-May-2019].
- [4] M. Duang, Victor. Teng, "Parallelized Sudoku Solver on the GPU," 2015. [Online]. Available: <https://github.com/vduan/parallel-sudoku-solver#summary>.

HPC Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

39/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

GRADING FORM: SMU GENERIC_GJ

STEFFAN BENSON

37.50

RESEARCH-INFORMED LITERATURE



Avoid refs in headings. Abstract a little long.
Overreliance on online refs (all of them).

5.5

KNOWLEDGE AND UNDERSTANDING



A good understanding of approaches to solving the puzzle. Historical context provided. Nothing on CUDA but parallelisation of the method is discussed- a more detailed discussion would help here.

8

ANALYSIS



Backtracking discussed as a method for solution and ref'd. Both linear (CPU based) and parallel method (GPU).

3.5

PRACTICAL APPLICATION AND DEP.



Limited - no results provided. An attempt made to convert this into a working solution - didn't produce any useful results for evaluation.

18

SKILLS FOR PROFESSIONAL PRAC.



Diagrams helpful; report structured with refs - avoid online refs.

2.5