Atividade 1 - Solucionador de Sudoku

Ekistoclecio Heleno - EHDL
Ian Torres - IKTS
Samuel Simões - SSSF2

Centro de Informática UFPE
Universidade Federal de Pernambuco

Aplicação e Medição de desempenho

- A aplicação consiste em um solucionador de sudoku utilizando a técnica backtracking e inicialização de validação aleatória.
    - O algoritmo utilizado começa a preencher em uma posição do tabuleiro com um valor aleatório e em seguida confere se os possíveis resultados preenchendo a tabela com novos valores geram um solução válida para o sudoku.
- Para medir o desempenho foi calculado o tempo gasto para encontrar uma solução válida para uma determinado configuração do tabuleiro sudoku.
    - Para evitar que configurações diferentes de tabuleiro tivessem impacto nos testes foram utilizados 3 modelos diferentes para o processo de análise.

Implementação - Solucionador

```
func solve(wg *sync.WaitGroup, id int) {
    defer wg.Done()

    size := 9

    board := setUpBoard(size, []int{3, 0, 6, 5, 0, 8, 4, 0, 0, 5, 2, 0, 0, 0, 0, 0, 0, 0, 0, 8,

    i, j, value := getRandomValues(size)
    // printInitial(i, j, value, id)

    start := time.Now()

    if solveRecursive(i, j, board, 0, value, size) {
        // printBoard(board, id, size)
        end := time.Now()
        printTime(start, end)
        if !finished {
            finished = true
        }
    }
}
```

Implementação - Solucionador recursivo

```
func solveRecursive(row, col int, board [][]int, xTimes, startV int, size int) bool {
    if xTimes == size*size {
        return true
    }

    isFull := true
    for p := 0; p < size; p++ {
        for q := 0; q < size; q++ {
            if board[p][q] == 0 {
                isFull = false
            }
        }
    }
    if isFull {
        return true
    }

    if col++; col == size {
        col = 0
        if row++; row == size {
            row = 0
        }
    }
    if board[row][col] != 0 {
        return solveRecursive(row, col, board, xTimes+1, startV, size)
    }

    for val := 1; val <= size; val++ {
        if startV++; startV == size+1 {
            startV = 1
        }

        if functions.ValidateSudoku(board, row, col, startV, size) {
            board[row][col] = startV
            if solveRecursive(row, col, board, xTimes+1, startV, size) {
                return true
            }
        }
    }

    board[row][col] = 0
    return false
}
```

Análise de Desempenho - Sistema

- Especificações do Sistema:
  - Processador: Intel(R) Core (TM) i7-1165G7 @ 2.80GHz, 4 núcleos, 8 processadores lógicos
  - Memória: 16GB
  - Sistema operacional: Linux Mint 21.1
  - Linguagem de programação: Go (Versão 1.21.1)
  - Interfaces de rede: Ligadas
  - Fonte de alimentação: Rede elétrica

Análise de Desempenho - Sistema

- Serviços do sistema:
  - Browser (Google Chrome)
  - Discord (com transmissão ao vivo)
  - Editor de texto (VSCode)

Análise de Desempenho - Métrica

A métrica utilizada é o tempo de duração da thread mais eficiente para a solução do algoritmo. No caso da execução single thread, a única thread existente, em um caso de multi thread, a que solucionar mais rápido.

Escolher a técnica de avaliação

- A técnica de avaliação consiste na medição do tempo médio gasto na resolução de três sudokus diferentes mil vezes cada um, utilizando uma thread, duas threads e cinco threads. Analisando o tempo médios de resolução dos sudokus para cada um dos três casos, é possível comparar o desempenho para cada um dos cenários.

Análise de Desempenho - Parâmetros

- Foram testadas 3 abordagens:
  - 1 thread
  - 2 threads
  - 5 threads

Análise de Desempenho - Carga

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

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

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

Análise de Desempenho - Projeto do experimento

```go
func main() {

    numTries := 1000

    for i := 0; i < numTries; i++ {
        run()
    }

    fmt.Println(times)
    fmt.Printf("%vµs\n", utils.GetMean(times))

}
```

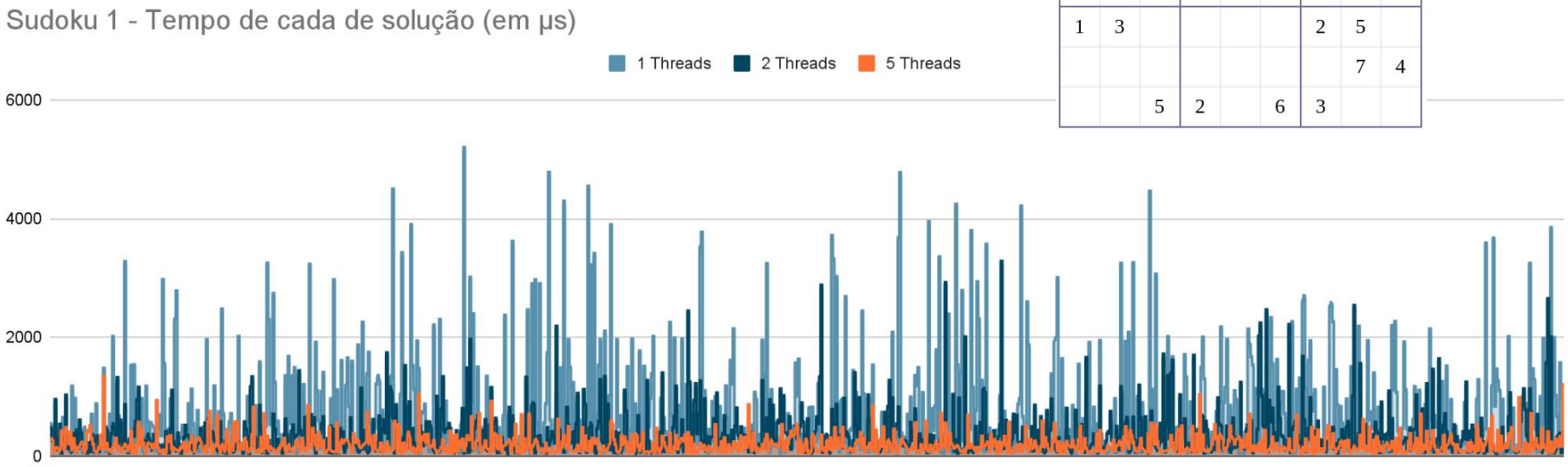
```go
func GetMean(values []int64) float64 {
    total := 0.0

    for _, value := range values {
        total += float64(value)
    }

    if len(values) == 0 {
        return 0.0 // Evita a divisão por zero
    }

    return total / float64(len(values))
}
```

Análise de Desempenho - Projeto do experimento

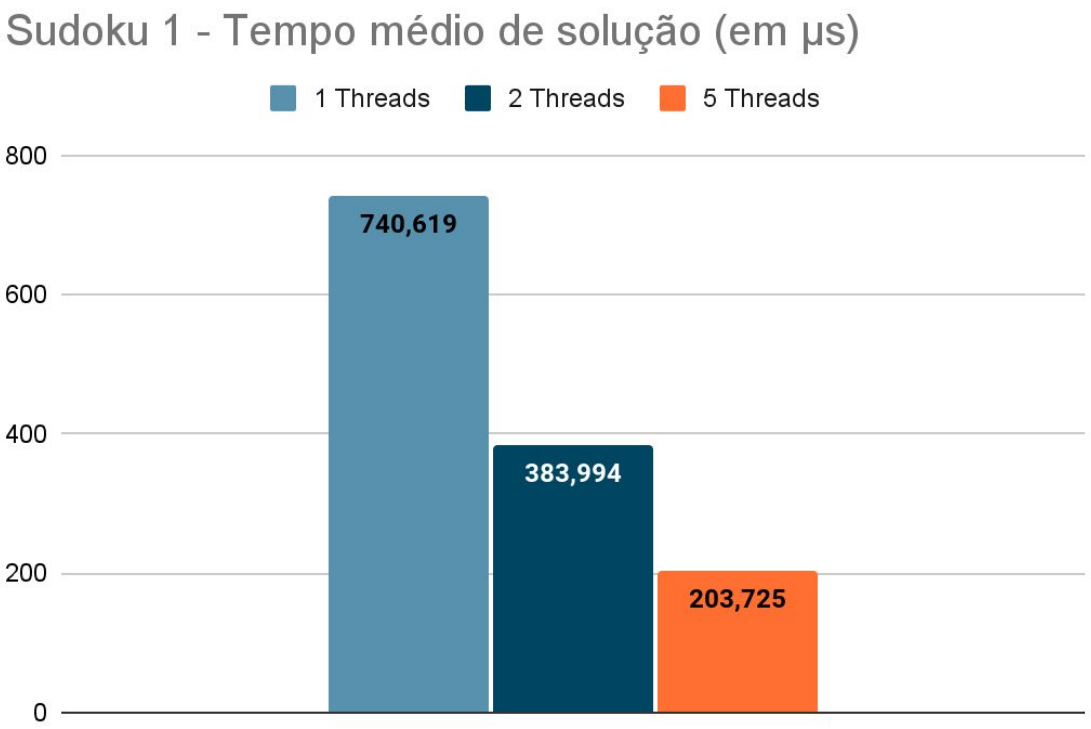
```go
func run() {
    numRoutines := 1
    //fmt.Printf("Using %d goroutines\n", numRoutines)

    finished = false
    s = 1
    var wg sync.WaitGroup
    wg.Add(numRoutines)

    for i := 0; i < numRoutines; i++ {
        go solve(&wg, i)
    }

    wg.Wait()
}
```

Análise de Desempenho

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



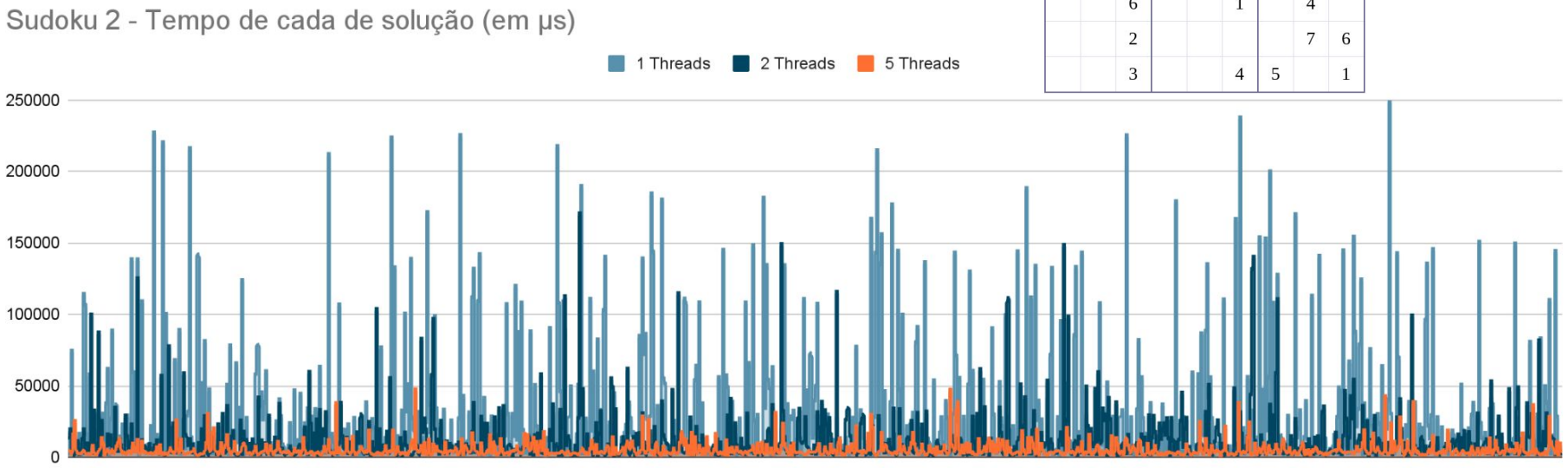
Análise de Desempenho



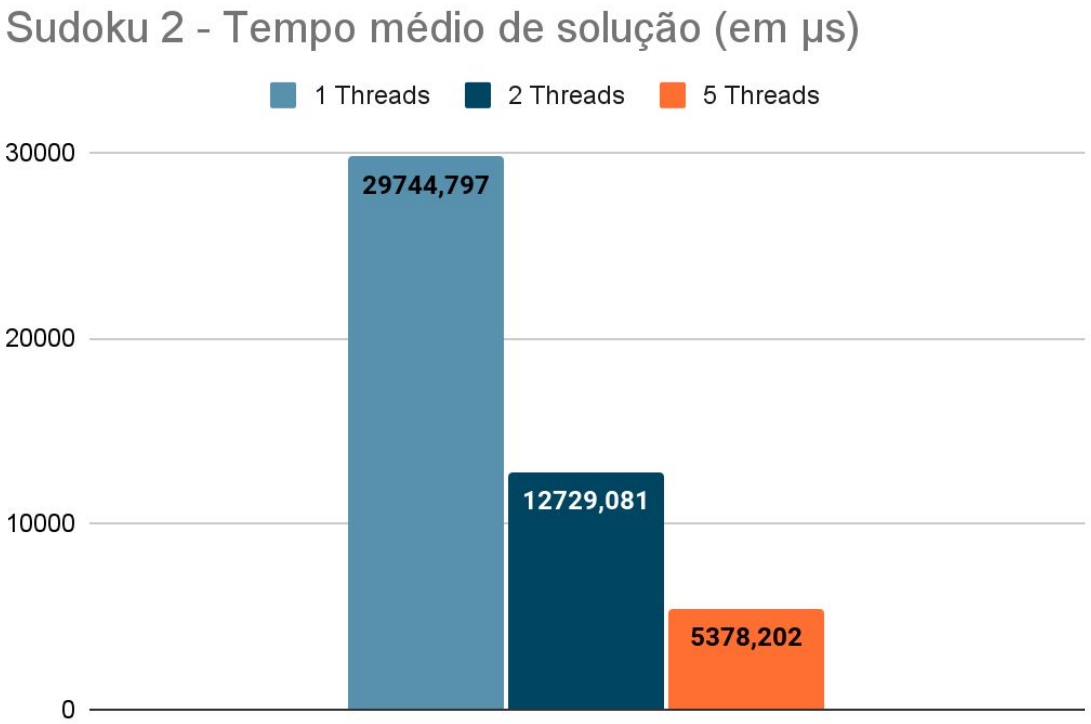
3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Análise de Desempenho

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



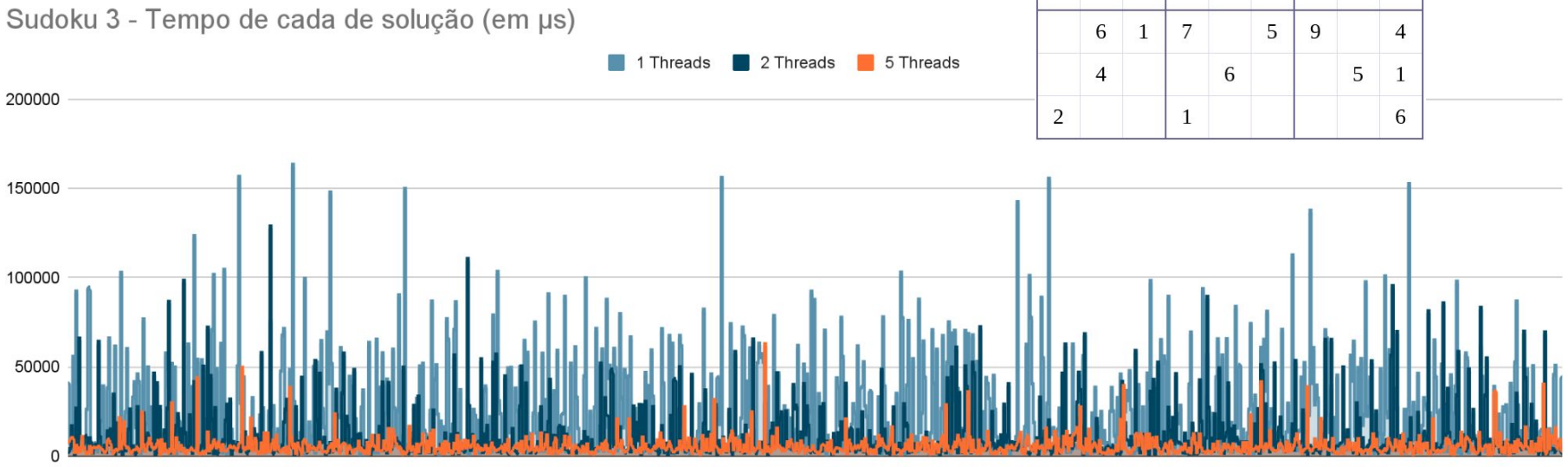
Análise de Desempenho



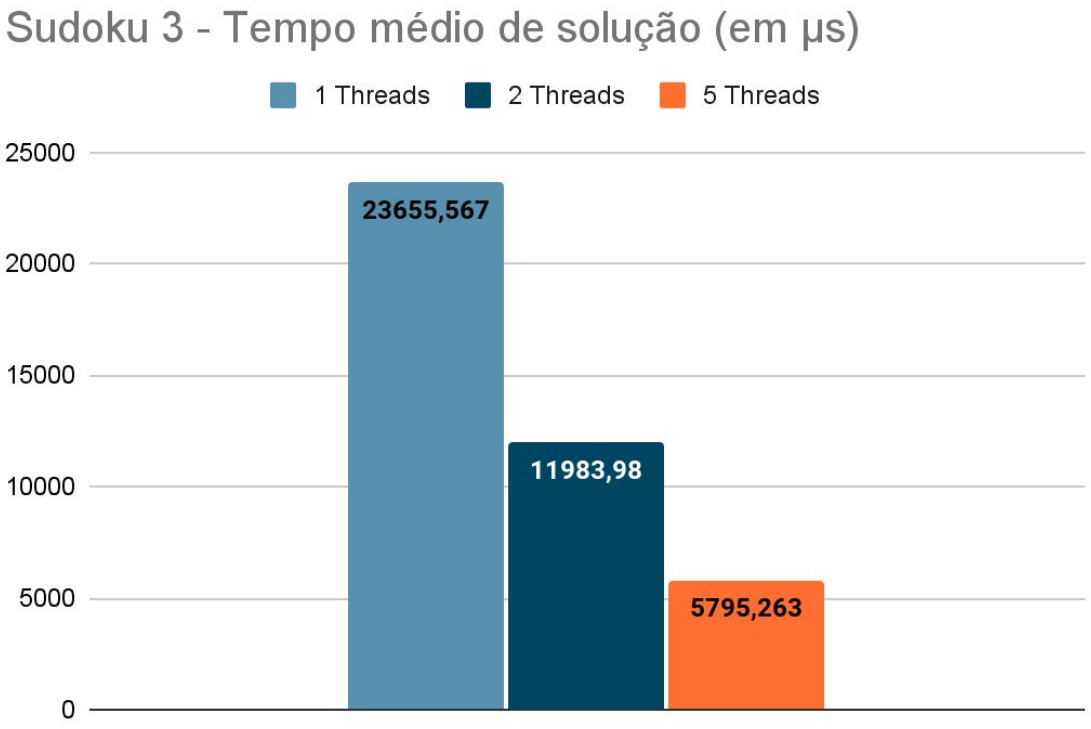
					5		6	
		5		6				
	7					4		
	8		1				2	
	3			7				
			5		2	1		3
		6			1		4	
		2					7	6
		3			4	5		1

Análise de Desempenho

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



Análise de Desempenho



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

Resultados

É possível observar que o aumento no número de threads utilizadas na resolução do sudoku gera um ganho significativo de desempenho resultando num tempo médio de solução até seis vezes menor ao utilizar cinco threads na solução em comparação com a utilização de uma única thread, logo pode-se afirmar que o uso de múltiplas thread com o intuito que agilizar o processo de resolução de um tabuleiro de sudoku foi bem sucedido.