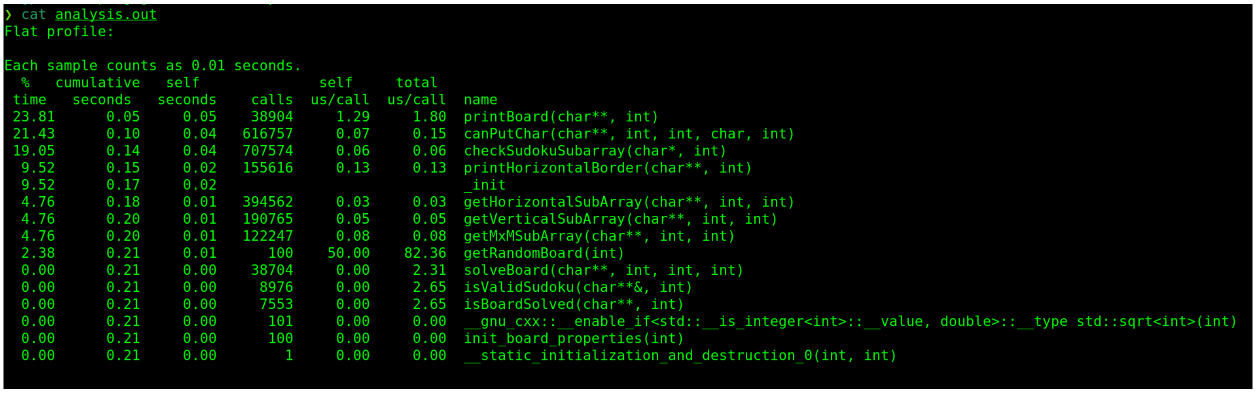
High Performance Computing
Week 3 - Profiling
Generation and Solving of NxN Sudoku Puzzles

Kotamarthi Mohan Himanshu

CED19I026

Functional profiling - gprof

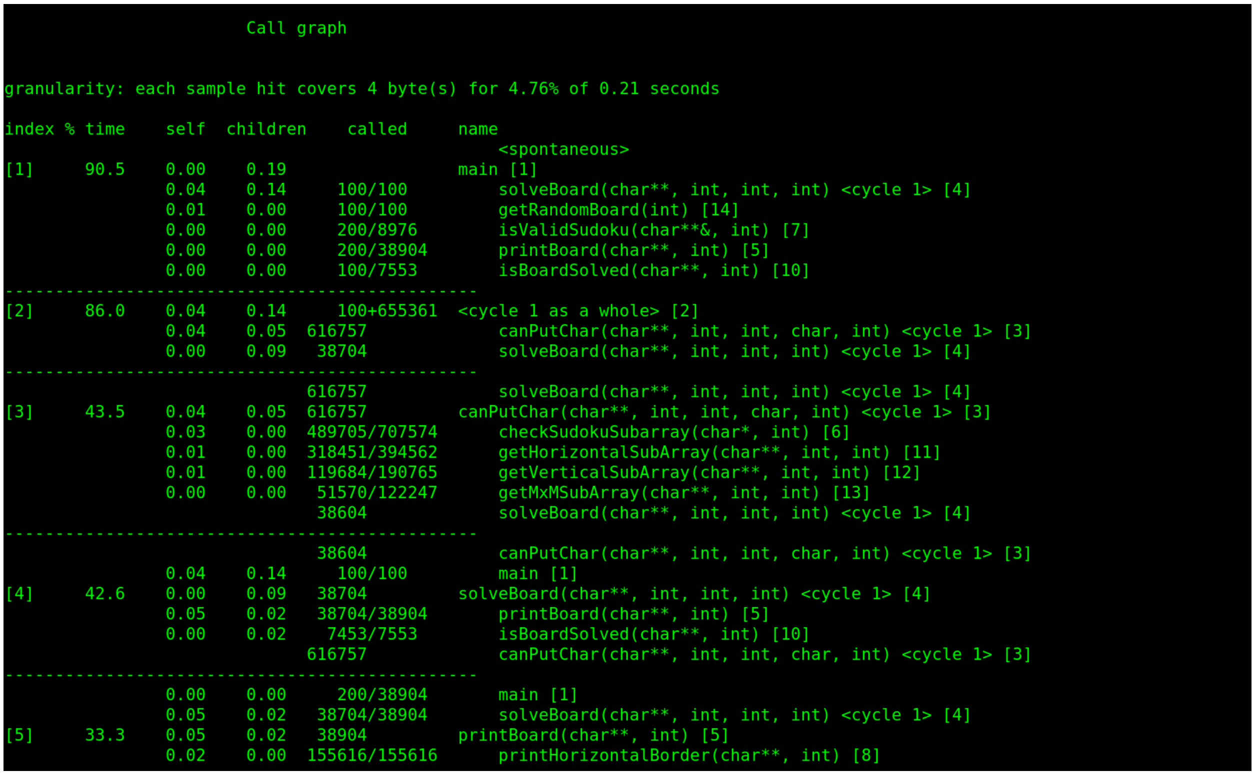
Flat Profile:



```
❯ cat analysis.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 23.81      0.05     0.05    38904     1.29     1.80  printBoard(char**, int)
 21.43      0.10     0.04   616757     0.07     0.15  canPutChar(char**, int, int, char, int)
 19.05      0.14     0.04   707574     0.06     0.06  checkSudokuSubarray(char*, int)
  9.52      0.15     0.02   155616     0.13     0.13  printHorizontalBorder(char**, int)
  9.52      0.17     0.02                             _init
  4.76      0.18     0.01   394562     0.03     0.03  getHorizontalSubArray(char**, int, int)
  4.76      0.20     0.01   190765     0.05     0.05  getVerticalSubArray(char**, int, int)
  4.76      0.20     0.01   122247     0.08     0.08  getMxMSubArray(char**, int, int)
  2.38      0.21     0.01      100    50.00    82.36  getRandomBoard(int)
  0.00      0.21     0.00    38704     0.00     2.31  solveBoard(char**, int, int, int)
  0.00      0.21     0.00     8976     0.00     2.65  isValidSudoku(char**&, int)
  0.00      0.21     0.00     7553     0.00     2.65  isBoardSolved(char**, int)
  0.00      0.21     0.00      101     0.00     0.00  __gnu_cxx::__enable_if<std::__is_integer<int>::__value, double>::__type std::sqrt<int>(int)
  0.00      0.21     0.00      100     0.00     0.00  init_board_properties(int)
  0.00      0.21     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
```

Call Graph:



```
                        Call graph


granularity: each sample hit covers 4 byte(s) for 4.76% of 0.21 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     90.5    0.00    0.19                 main [1]
                0.04    0.14     100/100         solveBoard(char**, int, int, int) <cycle 1> [4]
                0.01    0.00     100/100         getRandomBoard(int) [14]
                0.00    0.00     200/8976        isValidSudoku(char**&, int) [7]
                0.00    0.00     200/38904       printBoard(char**, int) [5]
                0.00    0.00     100/7553        isBoardSolved(char**, int) [10]
-----------------------------------------------
[2]     86.0    0.04    0.14     100+655361  <cycle 1 as a whole> [2]
                0.04    0.05  616757             canPutChar(char**, int, int, char, int) <cycle 1> [3]
                0.00    0.09   38704             solveBoard(char**, int, int, int) <cycle 1> [4]
-----------------------------------------------
                              616757             solveBoard(char**, int, int, int) <cycle 1> [4]
[3]     43.5    0.04    0.05  616757         canPutChar(char**, int, int, char, int) <cycle 1> [3]
                0.03    0.00  489705/707574      checkSudokuSubarray(char*, int) [6]
                0.01    0.00  318451/394562      getHorizontalSubArray(char**, int, int) [11]
                0.01    0.00  119684/190765      getVerticalSubArray(char**, int, int) [12]
                0.00    0.00   51570/122247      getMxMSubArray(char**, int, int) [13]
                               38604             solveBoard(char**, int, int, int) <cycle 1> [4]
-----------------------------------------------
                               38604             canPutChar(char**, int, int, char, int) <cycle 1> [3]
                0.04    0.14     100/100         main [1]
[4]     42.6    0.00    0.09   38704         solveBoard(char**, int, int, int) <cycle 1> [4]
                0.05    0.02   38704/38904       printBoard(char**, int) [5]
                0.00    0.02    7453/7553        isBoardSolved(char**, int) [10]
                              616757             canPutChar(char**, int, int, char, int) <cycle 1> [3]
-----------------------------------------------
                0.00    0.00     200/38904       main [1]
                0.05    0.02   38704/38904       solveBoard(char**, int, int, int) <cycle 1> [4]
[5]     33.3    0.05    0.02   38904         printBoard(char**, int) [5]
                0.02    0.00  155616/155616      printHorizontalBorder(char**, int) [8]
```

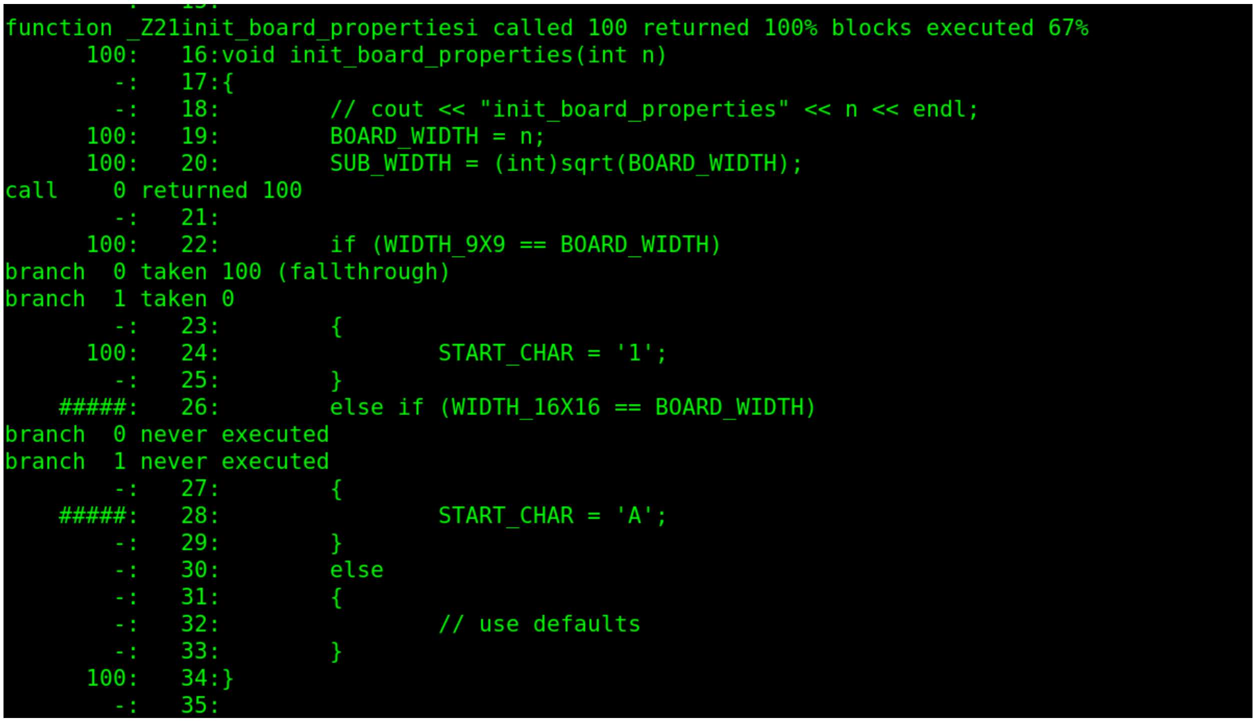
From this we get to know that almost 60% of the execution time is spent in the top 3 functions.

- printBoard()
- canPutChar()
- checkSudokuSubarray()

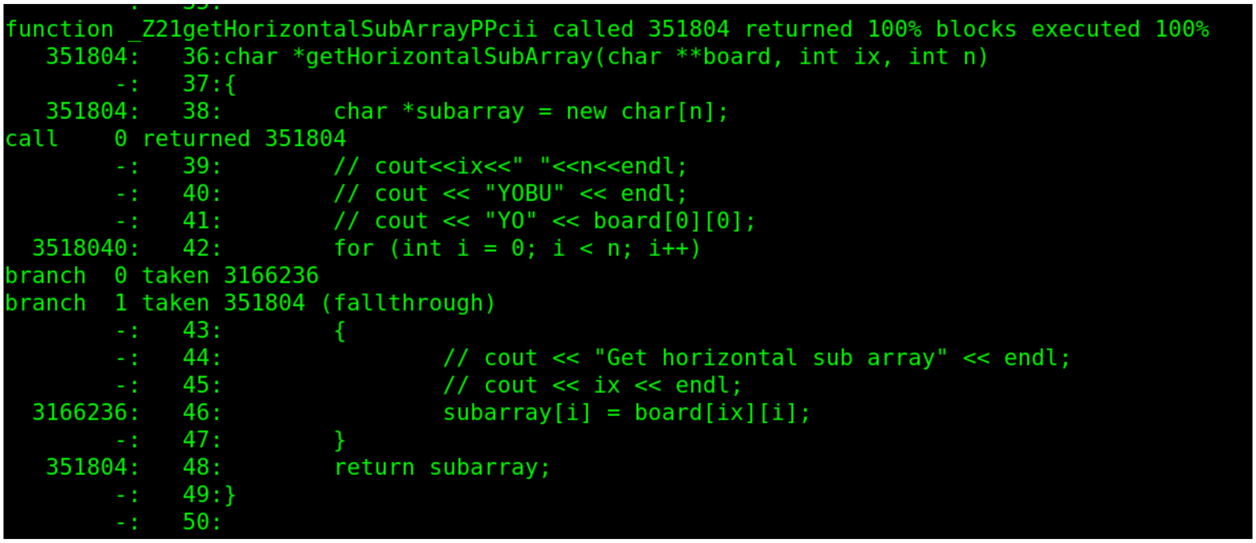
printBoard() can’t be parallelized,so we have to reduce the amount of times its called in the program.
So, if we are able to parallelize these functions,we will be able to reduce the execution time by a large amount.

Line Profiling - gcov

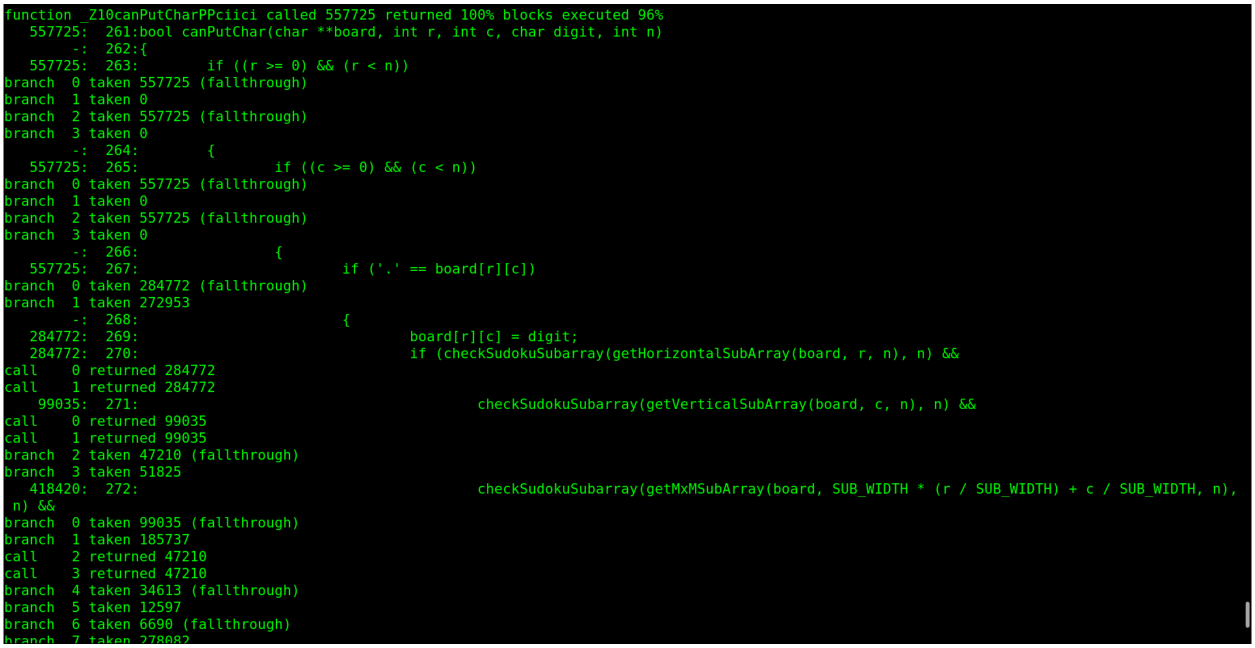
Gcov is a source code coverage analysis and statement-by-statement profiling tool. Gcov generates exact counts of the number of times each statement in a program is executed and annotates source code to add instrumentation.



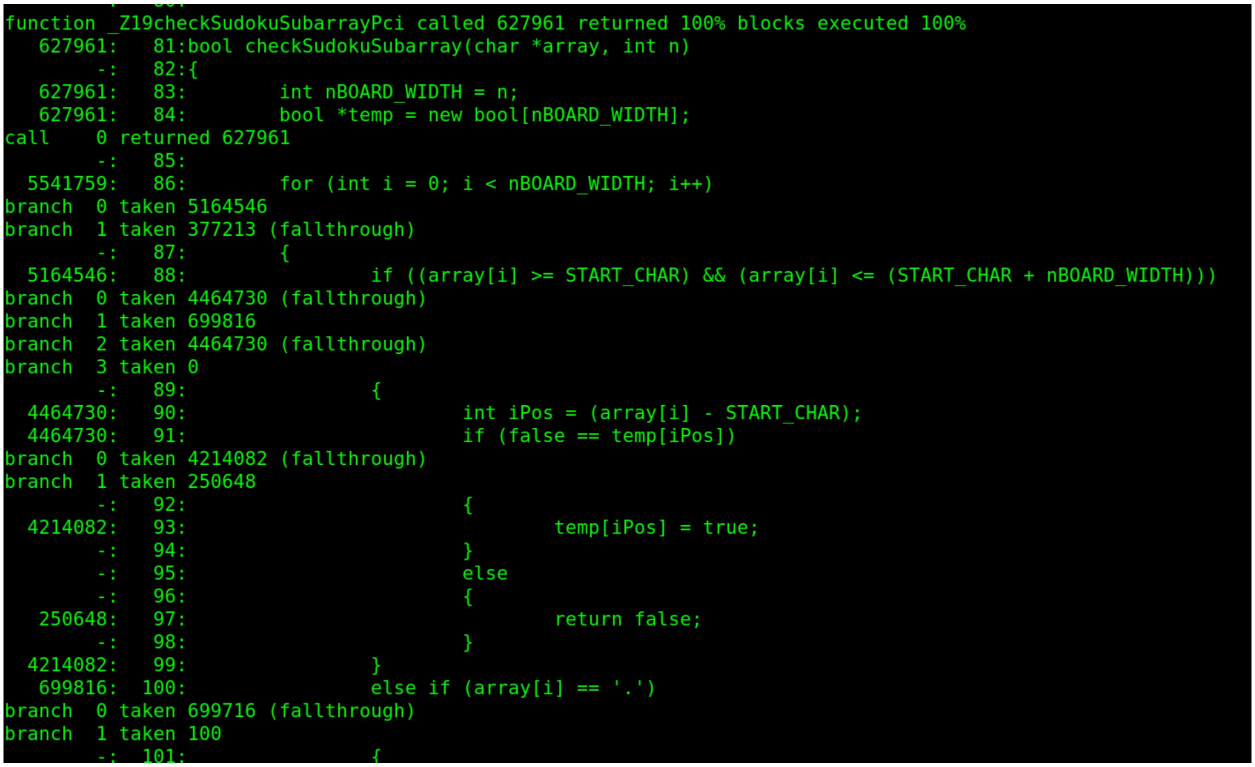
```
        -:   15:
function _Z21init_board_propertiesi called 100 returned 100% blocks executed 67%
      100:   16:void init_board_properties(int n)
        -:   17:{
        -:   18:        // cout << "init_board_properties" << n << endl;
      100:   19:        BOARD_WIDTH = n;
      100:   20:        SUB_WIDTH = (int)sqrt(BOARD_WIDTH);
call    0 returned 100
        -:   21:
      100:   22:        if (WIDTH_9X9 == BOARD_WIDTH)
branch  0 taken 100 (fallthrough)
branch  1 taken 0
        -:   23:        {
      100:   24:                START_CHAR = '1';
        -:   25:        }
    #####:   26:        else if (WIDTH_16X16 == BOARD_WIDTH)
branch  0 never executed
branch  1 never executed
        -:   27:        {
    #####:   28:                START_CHAR = 'A';
        -:   29:        }
        -:   30:        else
        -:   31:        {
        -:   32:                // use defaults
        -:   33:        }
      100:   34:}
        -:   35:
```



```
        -:   35:
function _Z21getHorizontalSubArrayPPcii called 351804 returned 100% blocks executed 100%
   351804:   36:char *getHorizontalSubArray(char **board, int ix, int n)
        -:   37:{
   351804:   38:        char *subarray = new char[n];
call    0 returned 351804
        -:   39:        // cout<<ix<<" "<<n<<endl;
        -:   40:        // cout << "YOBU" << endl;
        -:   41:        // cout << "YO" << board[0][0];
  3518040:   42:        for (int i = 0; i < n; i++)
branch  0 taken 3166236
branch  1 taken 351804 (fallthrough)
        -:   43:        {
        -:   44:                // cout << "Get horizontal sub array" << endl;
        -:   45:                // cout << ix << endl;
  3166236:   46:                subarray[i] = board[ix][i];
        -:   47:        }
   351804:   48:        return subarray;
        -:   49:}
        -:   50:
```



```
function _Z10canPutCharPPciici called 557725 returned 100% blocks executed 96%
   557725:  261:bool canPutChar(char **board, int r, int c, char digit, int n)
        -:  262:{
   557725:  263:        if ((r >= 0) && (r < n))
branch  0 taken 557725 (fallthrough)
branch  1 taken 0
branch  2 taken 557725 (fallthrough)
branch  3 taken 0
        -:  264:        {
   557725:  265:                if ((c >= 0) && (c < n))
branch  0 taken 557725 (fallthrough)
branch  1 taken 0
branch  2 taken 557725 (fallthrough)
branch  3 taken 0
        -:  266:                {
   557725:  267:                        if ('.' == board[r][c])
branch  0 taken 284772 (fallthrough)
branch  1 taken 272953
        -:  268:                        {
   284772:  269:                                board[r][c] = digit;
   284772:  270:                                if (checkSudokuSubarray(getHorizontalSubArray(board, r, n), n) &&
call    0 returned 284772
call    1 returned 284772
    99035:  271:                                        checkSudokuSubarray(getVerticalSubArray(board, c, n), n) &&
call    0 returned 99035
call    1 returned 99035
branch  2 taken 47210 (fallthrough)
branch  3 taken 51825
   418420:  272:                                        checkSudokuSubarray(getMxMSubArray(board, SUB_WIDTH * (r / SUB_WIDTH) + c / SUB_WIDTH, n),
 n) &&
branch  0 taken 99035 (fallthrough)
branch  1 taken 185737
call    2 returned 47210
call    3 returned 47210
branch  4 taken 34613 (fallthrough)
branch  5 taken 12597
branch  6 taken 6690 (fallthrough)
branch  7 taken 278082
```



```
        -:   80:
function _Z19checkSudokuSubarrayPci called 627961 returned 100% blocks executed 100%
   627961:   81:bool checkSudokuSubarray(char *array, int n)
        -:   82:{
   627961:   83:        int nBOARD_WIDTH = n;
   627961:   84:        bool *temp = new bool[nBOARD_WIDTH];
call    0 returned 627961
        -:   85:
  5541759:   86:        for (int i = 0; i < nBOARD_WIDTH; i++)
branch  0 taken 5164546
branch  1 taken 377213 (fallthrough)
        -:   87:        {
  5164546:   88:                if ((array[i] >= START_CHAR) && (array[i] <= (START_CHAR + nBOARD_WIDTH)))
branch  0 taken 4464730 (fallthrough)
branch  1 taken 699816
branch  2 taken 4464730 (fallthrough)
branch  3 taken 0
        -:   89:                {
  4464730:   90:                        int iPos = (array[i] - START_CHAR);
  4464730:   91:                        if (false == temp[iPos])
branch  0 taken 4214082 (fallthrough)
branch  1 taken 250648
        -:   92:                        {
  4214082:   93:                                temp[iPos] = true;
        -:   94:                        }
        -:   95:                        else
        -:   96:                        {
   250648:   97:                                return false;
        -:   98:                        }
  4214082:   99:                }
   699816:  100:                else if (array[i] == '.')
branch  0 taken 699716 (fallthrough)
branch  1 taken 100
        -:  101:                {
```

From this we understand that there are parts of the code which are never executed.These are some initialization conditions.

There are also parts of the code which are executed more than others.Within the top 3 functions, there are loops that can be collapsed and parallelized.There are some conditions which are never met. Those can be removed from the code.This makes it more efficient, and results in better code density.

Hardware resource profiling- likwid

Output:

```
--------------------------------------------------------------------------------
CPU name:   11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz
CPU type:   Intel Tigerlake processor
CPU stepping:        2
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:             1
Cores per socket:    4
Threads per core:    2
--------------------------------------------------------------------------------
HWThread    Thread              Core          Socket                 Available
0           0           0           0           *
1           0           1           0           *
2           0           2           0           *
3           0           3           0           *
4           1           0           0           *
5           1           1           0           *
6           1           2           0           *
7           1           3           0           *
--------------------------------------------------------------------------------
Socket 0:            ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:               1
Size:                48 kB
Cache groups:               ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:               2
```

```
Size:                   1 MB
Cache groups:           ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   8 MB
Cache groups:           ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           1
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 4 1 5 2 6 3 7 )
Distances:              10
Free memory:            10875.6 MB
Total memory:           15752.9 MB
--------------------------------------------------------------------------------


********************************************************************************
Graphical Topology
********************************************************************************
Socket 0:
+-----------------------------------------+
| +--------+ +--------+ +--------+ +--------+ |
| |  0 4  | |  1   5 | |  2   6 | |  3   7 | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| | 48 kB | | 48 kB | | 48 kB | | 48 kB | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| |  1 MB | |  1 MB | |  1 MB | |  1 MB | |
| +--------+ +--------+ +--------+ +--------+ |
| +-------------------------------------+ |
| |               8 MB               | |
| +-------------------------------------+ |
+-----------------------------------------+
```

Inference:

Using gprof we have found out that the top 3 functions in terms of execution time are printBoard(), canPutChar() and checkSudokuSubarray().Within these functions we have found which lines are executed the most using gcov.This has given us a good idea of the exact areas to focus on to improve the execution time of the program.

Likwid also has given us an idea of the hardware topology of our computer.This will give us an idea on how to utilize the shared memory and variables when parallelize the program.