CO21BTECH11002

Aayush Kumar

OS2 Assignment 2

The checkValid( ) function:

We first initialize the freq array for storing the frequency of the points.

This function takes the struct values as input.

For each row/column/grid we calculate the number of times each value appears by storing its frequency in the freq array.

If we get an invalid number in sudoku (like negative number or number bigger than size of side of sudoku) we set flag equal to zero which indicates that the row/column/grid is invalid.

Similarly, if a point is found to appear more than once we set flag equal to zero.

We continue this processes for all the rows/columns/grids that the thread needs to check.

The main( ) function:

We start by taking reading the value of n and k from the input file.

The program initializes the number of rows/columns/grids each thread needs to check (at max n/k + 1).

For pthreads:

It creates and runs all the threads and waits for them to finish. After the threads have executed, it joins all the threads.

For OpenMP:

It starts the parallel region which is then handled by OpenMP library.

Then it creates the Output file and writes in it using the values found by running the threads, stored in threadValues array.

Results:

After running the program 5 times and taking the average value of time taken (in microseconds), we get the following results:

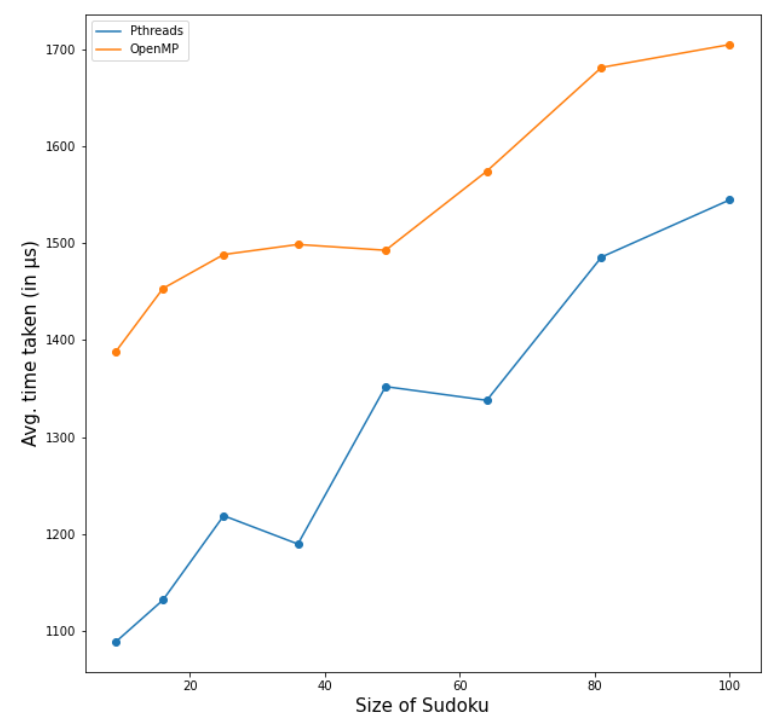
Time taken (in microseconds) vs size of sudoku (keeping no. of threads = 16):

For Pthreads:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Points | Exec 1 | Exec 2 | Exec 3 | Exec 4 | Exec 5 | Average |
| 9x9 | 909 | 1179 | 1124 | 1177 | 1054 | 1088.6 |
| 16x16 | 1155 | 1140 | 1068 | 1196 | 1100 | 1131.8 |
| 25x25 | 1256 | 1385 | 1064 | 1140 | 1249 | 1218.8 |
| 36x36 | 1268 | 1186 | 1185 | 1102 | 1207 | 1189.6 |
| 49x49 | 1299 | 1519 | 1095 | 1554 | 1293 | 1352.0 |
| 64x64 | 1333 | 1273 | 1476 | 1210 | 1397 | 1337.8 |
| 81x81 | 1347 | 1390 | 1907 | 1377 | 1407 | 1485.6 |
| 100x100 | 1367 | 1337 | 1853 | 1619 | 1547 | 1544.6 |

For OpenMp:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Points | Exec 1 | Exec 2 | Exec 3 | Exec 4 | Exec 5 | Average |
| 9x9 | 1312 | 1401 | 1390 | 1393 | 1444 | 1388.0 |
| 16x16 | 1412 | 1509 | 1441 | 1462 | 1442 | 1453.2 |
| 25x25 | 1564 | 1473 | 1452 | 1516 | 1436 | 1488.2 |
| 36x36 | 1425 | 1461 | 1544 | 1602 | 1461 | 1498.6 |
| 49x49 | 1597 | 1429 | 1492 | 1439 | 1506 | 1492.6 |
| 64x64 | 1651 | 1458 | 1433 | 1853 | 1476 | 1574.2 |
| 81x81 | 1762 | 1681 | 1484 | 1658 | 1821 | 1681.2 |
| 100x100 | 1458 | 1898 | 1933 | 1490 | 1745 | 1704.8 |



We can see that although we get an almost increasing graph for both type of threads, Pthreads always performed better than OpenMP thread.

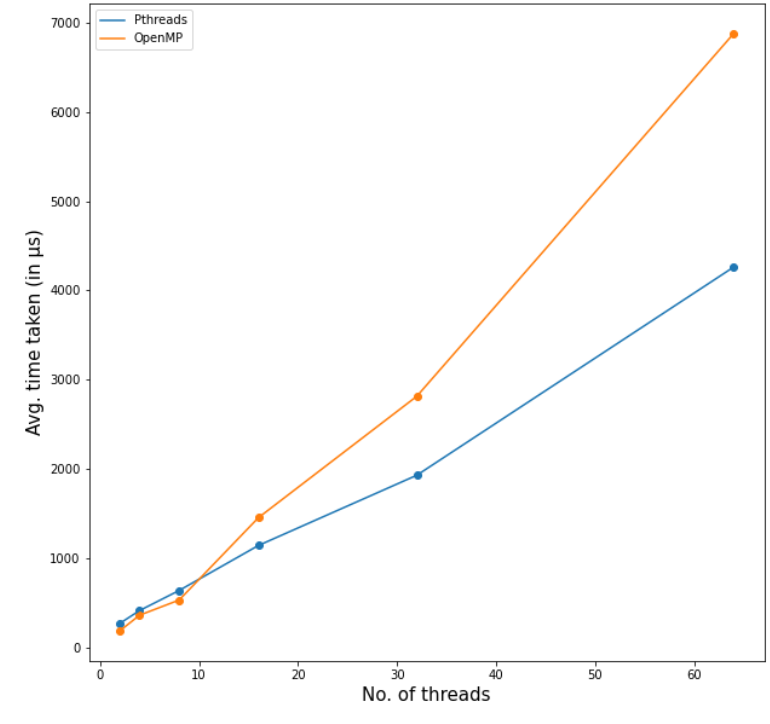
Time taken vs no. of threads (keeping size of sudoku = 25x25):

Pthreads:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Threads | Exec 1 | Exec 2 | Exec 3 | Exec 4 | Exec 5 | Average |
| 2 | 277 | 281 | 273 | 286 | 268 | 277.0 |
| 4 | 401 | 394 | 410 | 372 | 529 | 421.2 |
| 8 | 591 | 639 | 620 | 680 | 699 | 645.8 |
| 16 | 1050 | 1497 | 1090 | 1138 | 968 | 1148.6 |
| 32 | 1994 | 1918 | 1801 | 1868 | 2091 | 1934.4 |
| 64 | 4276 | 4128 | 4244 | 4805 | 3859 | 4262.4 |

For OpenMp:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Threads | Exec 1 | Exec 2 | Exec 3 | Exec 4 | Exec 5 | Average |
| 2 | 227 | 198 | 171 | 186 | 186 | 193.6 |
| 4 | 364 | 338 | 396 | 365 | 383 | 369.2 |
| 8 | 700 | 464 | 439 | 576 | 516 | 539.0 |
| 16 | 1699 | 1393 | 1364 | 1421 | 1420 | 1459.4 |
| 32 | 3201 | 2827 | 2921 | 2590 | 2548 | 2817.4 |
| 64 | 6726 | 7320 | 7185 | 6117 | 7020 | 6873.6 |



Here, the overhead of creating the threads is much higher than the actual computation time taken by threads. Hence, we get an increasing graph. But here too we can see that for bigger number of threads, Pthreads performed better since the overhead for creating OpenMP threads is much higher.