COMP 529 **Shukhrat Khuseynov (0070495)** **08.03.2020**

Assignment 1 - Part 2 (Report)

The task parallelism was implemented for the N-queens problem. All the parallel versions specified in the assignment description were completed and saved with relevant names: nqueens_parA.c, nqueens_parB.c, and nqueens_parA.c with nqueens_ser.c and nqueens_serC.c for the respective serial codes. Intermediate versions were not submitted, since there are too many versions anyways. For the easier submission of codes through batch file, N was inputted as a command line argument in all codes.

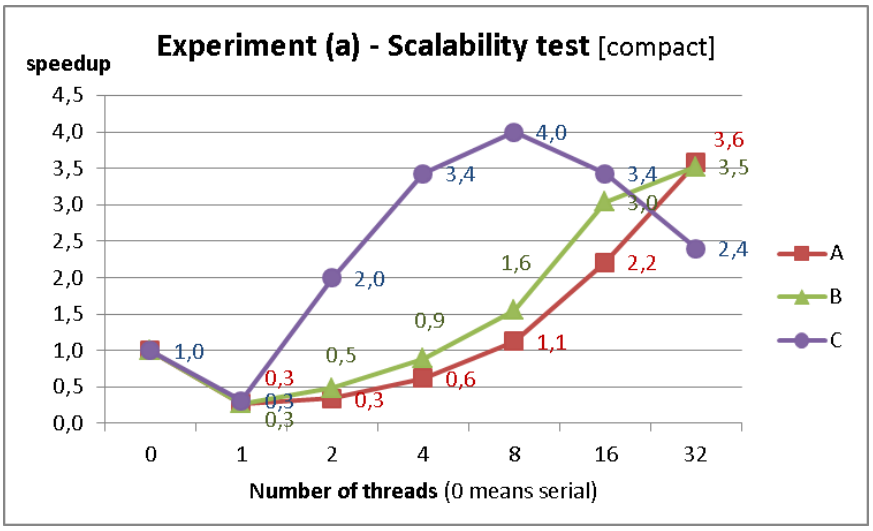
The task parallelism constituted mainly in allocating each iteration of the main for loop of N-queens recursion to an independent task. The taskwait was needed as a barrier after the end of the loop to end all tasks before returning from the function. Since return is not allowed within a parallel region, the function only returns after taskwait barrier. Other return points were modified to conditional statements within the function. The usage of SOLUTION_EXISTS boolean was optimized to eliminate some mutual exclusions (locks) of parallel region.

Within each generated task, several input parameters were specified as firstprivate. However, making board matrix firstprivate did not help, since it still pointed to the same memory space. This issue was resolved by creating a private buffer matrix within each task. The buffer matrix is initiated as equal to the input board matrix and then changes as private variable.

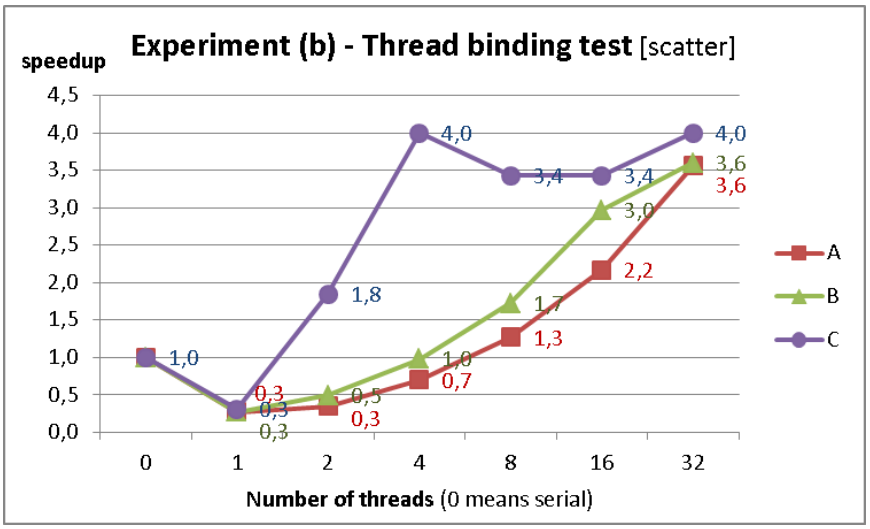
For the B part, creation of additional depth variable would create overhead, therefore the measure for depth was chosen as column (col in the code), which is similar to depth. After many runs and submissions to the KUACC cluster, the optimal number of first 6 columns was chosen to have tasks done in parallel, all the remaining would continue having tasks in serial. For the C part, making recursion stop after finding a solution was managed with conditional statements, using SOLUTION_EXISTS boolean. In the experiments of C part, since the asked N in the description is 14, which executes too fast - almost 0.00 seconds, for better calculations of speedup the N of 20 was chosen. Otherwise, numbers for speedup would be invalid.

The correctness of parallel versions was ensured by running many different inputs with smaller N, to be able to check, and comparing the output with the serial versions, assuming that the serial implementation is correct. The print_solution function was activated while doing that. Those checks showed identical results.

The experiments were conducted as asked in the assignment description. For the first two experiments, (a) and (b), they were run with three batch files: submit_jobA.sh, submit_jobB.sh, and submit_jobC.sh with outputs in nqueensA.out, nqueensB.out, nqueensC.out for the parts A, B, and C, respectively. The (c) experiment was run in submit_jobA_c.sh with nqueensA_c.out as output. I tried to run everything from one file, but the space of one *.out file is limited, everything does not fit. Here are the plotted graphs:

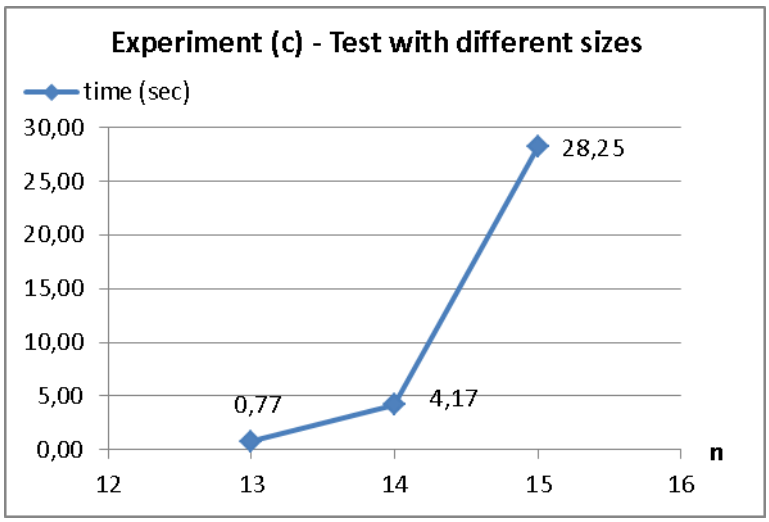


Experiment (a), being a strong scaling experiment, truly demonstrated a tendency of the speedup to increase with the numbers of threads, as seen above. For some reason the speedup of version C decreases starting with 16 threads, maybe due to some inefficiency in the code, which is designed not only for finding one solution but all or maybe parallelization overhead with some machine specificities.



Experiment (b), kept alone, also shows the increased speedup. Version C here operates better, maybe due to scatter mapping scheme. Moreover, if to compare compact and scatter schemes, it seems that as the number of threads increases, having scatter makes code work faster for all cases.

Experiment (c), being more similar to weak scaling studies, depicted how running time increases with larger input parameter of N. Its graph can be seen below:



In a nutshell, having also done many other runs, the noticeable speedup can be observed, which is rather fascinating. The task parallelism with recursion was a new thing I learned, doing this assignment.