PDE Solvers for the Laplace Equation

Introduction

The problem is to solve partial differential equations using grid computations that rely on iterative methods such as the Jacobi method and Gauss-Seidel method. This report will show how the programs (the ones listed in the instructions) are implemented and their performance evaluation for different numbers of grid sizes and threads.

Programs

I implemented the jacobi method and multigrid v-cycle method which are run serially. The parallel versions are parallelization of these serial programs.

The serial program based on jacobi method uses two grids for the iterations. The values in the grids are strictly calculated according to the formula provided in the instruction. Before the execution of the iterations the grids are also initialized in the exact way the instructions detail. The parameters for the program execution are also provided as command-line arguments. After the iterations, the maximum error is also calculated by comparing corresponding values in respective grids and taking the maximum difference (follows the pseudocode in FIgure 11.2).

The serial program based on the multigrid v-cycle method uses in total 8 grids for the iterations. The initialization of the grids is based on the instructions given. The interpolation and restriction operators strictly follow from the slides for lecture 18. When moving the levels in the v-cycle a mapping of 2:1 is used when calculating the new values for the grids on which the jacobi method will operate on afterwards. The rest of the program is identical to the serial program based on jacobi method.

To parallelize these serial programs, openMP was used. For the parallel program based on jacobi method, openMP directives for parallelizing loop iterations were added in the code. Each thread(worker) is guaranteed to have roughly the same amount of rows to process by using the schedule(static, _) clause and the strip size was calculated by dividing the gridSIze by the number of workers. For the parallel program based on multigrid v-cycle method the same approach was taken as the corresponding program based on jacobi method. Furthermore, the restriction operator, interpolation operator and max difference operator were also parallelized in the same manner as described above as they all had similar loop structures. In the max difference operator a custom reduction was declared that would make it easier to find the max difference. It was implemented so that no shared variable was necessary to be created. Instead every thread calculates the max difference in its strip and then the largest max difference is outputted as a result.

Performance Evaluation

These tests were performed on one of KTH:s computers that has 11th Generation Intel® Core™ i7 Processors with 8 cores and up to 16 threads. Every details outlined in the instruction for performance evaluation was followed.

Program1:

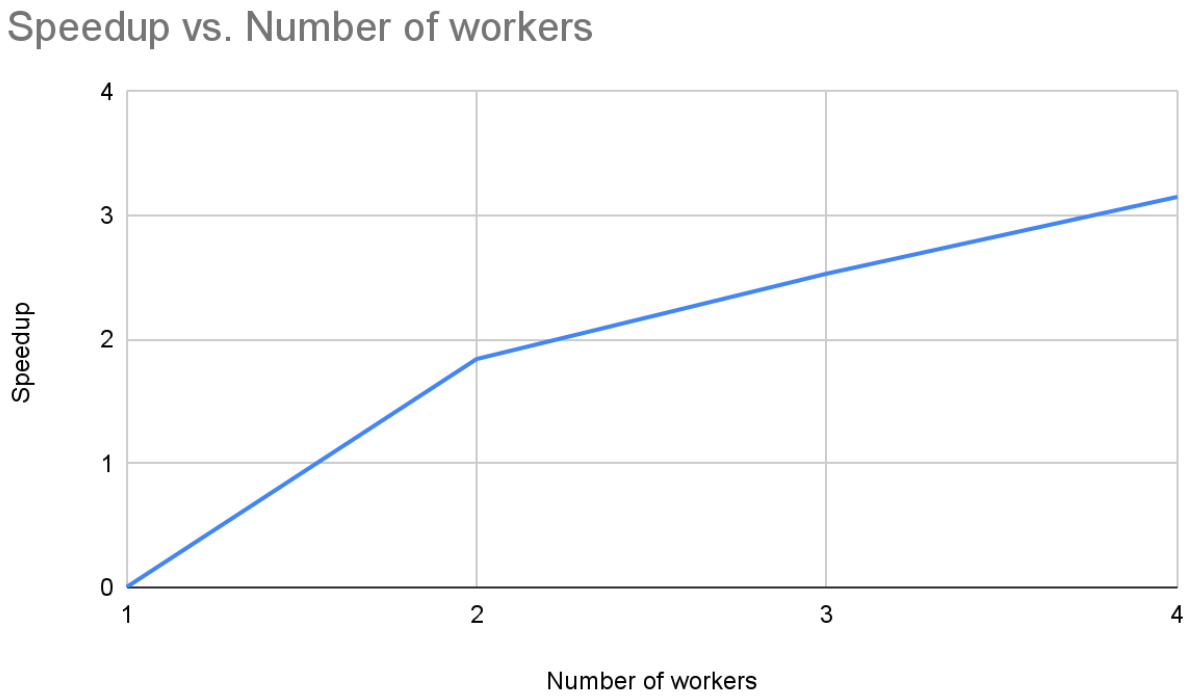
The number of iterations needed to make the execution time 30 seconds long is given for both grids.
gridSize = 100, numIters = 1000000 execution_time = 28.5597 sec
gridSize = 200, numIters = 250000 executioin_time = 28.416 sec

Program2:

The speedup is as expected even though it is a bit lower. This can depend on the high number of iterations that cause the parallel regions to be recreated as many times and cause the overhead.

gridSize = 100, numIters = 1000000

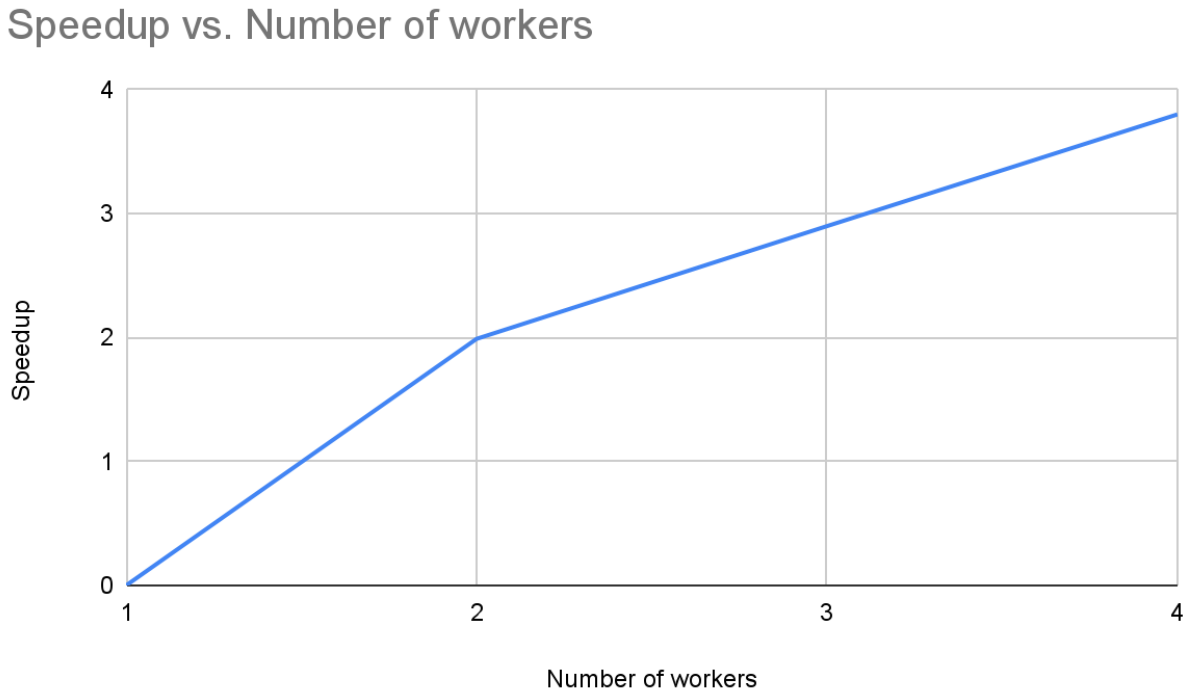
Number of workers	Execution-time	Speedup
1	28.7018 sec	
2	15.5773 sec	1.842540106
3	11.3355 sec	2.532027701
4	9.10883 sec	3.15098646



The speedup for this gridSize is much better and this can be due to the lower amount of iterations than the first gridSize. This means that there was a lower overhead to execute on this gridSize than on the first one and hence the better speedup.

gridSize = 200, numIters = 250000

Number of workers	Execution-time	Speedup
1	28.3855 sec	
2	14.2695 sec	1.989242791
3	9.79119 sec	2.899085811
4	7.46916 sec	3.800360415



Program3:

The number of iterations needed to make the execution time 30 seconds long is given for both grids.

gridSize = 12, numIters = 80 000 000 execution_time = 30.3342 sec
gridSize = 24, numIters = 20 000 000 executioin_time = 30.9818 sec

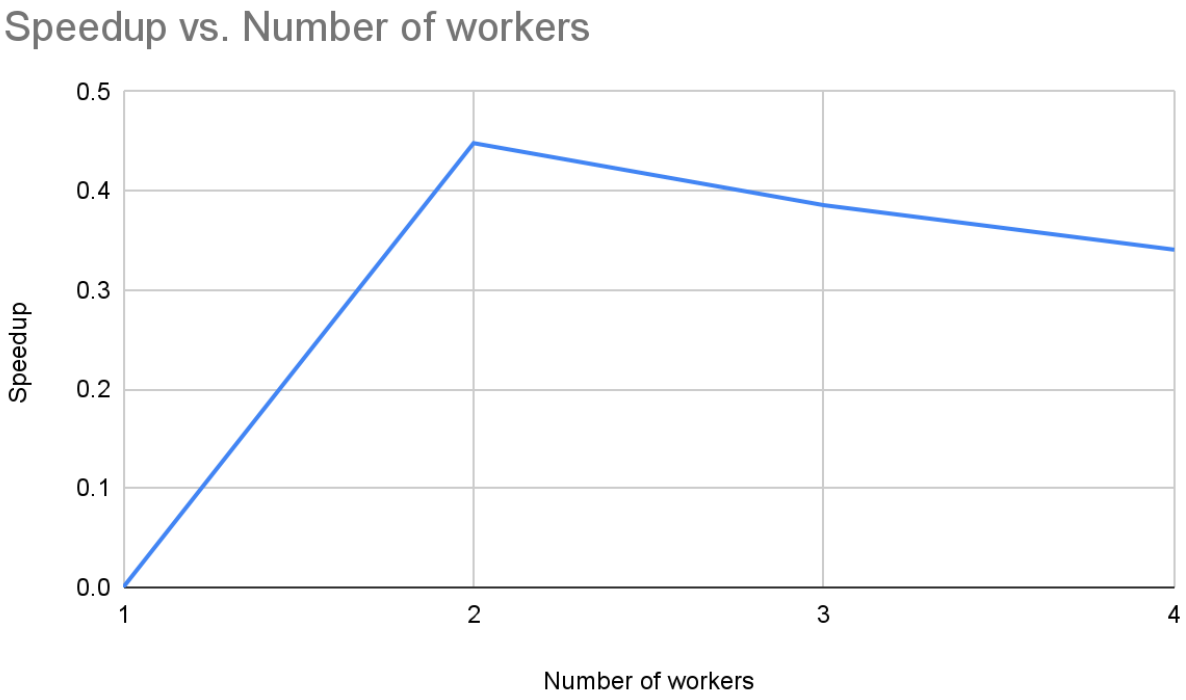
Program4:

In this program the jacobi-method, restriction operator and interpolation operator were parallelized. So that in itself incurs overhead in the execution time as there were nested for loops in all of these functions as well as barriers. Additionally the huge amount of iterations add

up onto the overhead and this results in a non-satisfactory speedup. The number of iterations were modified so as to make sure that the executions done with one thread last only 30 seconds.

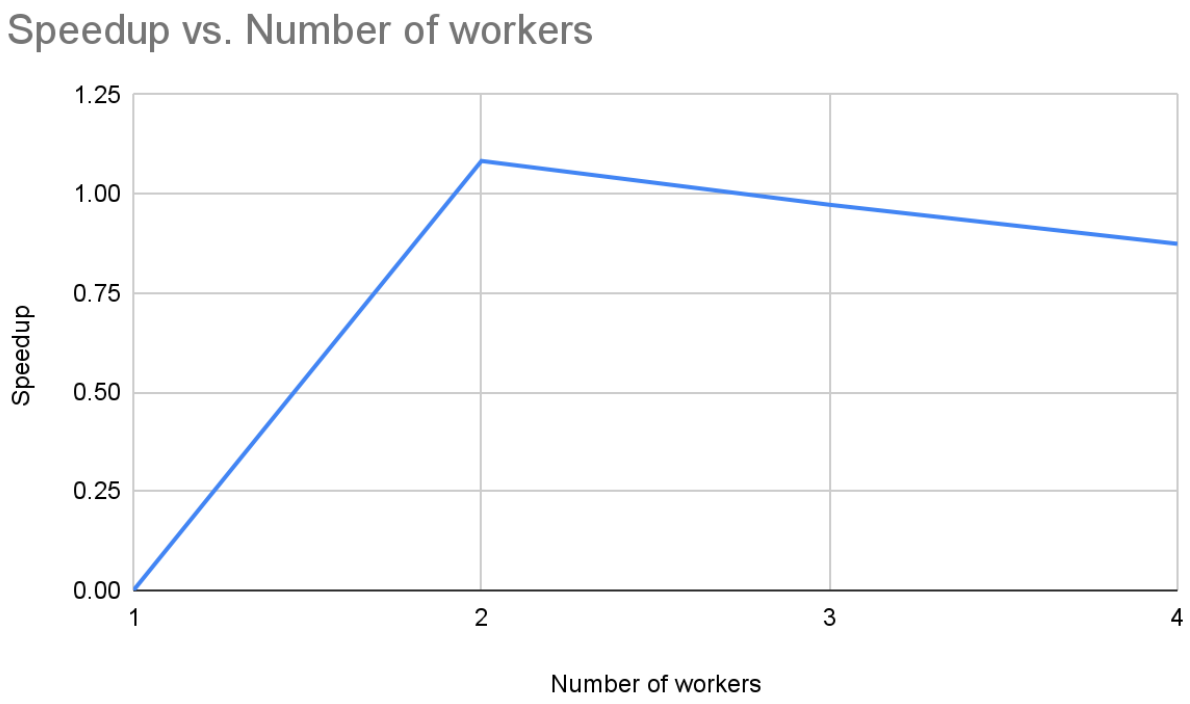
gridSize = 12, numIters = 40 000 000

Number of workers	Execution-time	Speedup
1	30.0085 sec	
2	66.9693 sec	0.44809338
3	77.8473 sec	0.385479008
4	88.1002 sec	0.340617842



gridSize = 24, numIters = 15 000 000

Number of workers	Execution-time	Speedup
1	31.1216 sec	
2	28.7416 sec	1.082806803
3	32.0044 sec	0.972416293
4	35.6117 sec	0.873915034



Conclusion

This report has shown that barriers and synchronization greatly affect the performance of parallel programs. Furthermore, the creation and destruction of threads repeatedly is also an overhead cost that significantly degrades performance of parallel programs and their scalability.

I have learned how data parallelism works as these programs are hugely dependent on the idea of doing the same operations on different sets of data. The problems that I faced when implementing the project was understanding how the restriction and interpolation operators in the multigrid v-cycle method work.