Assignment 03

Exercise 1:

a) Extend the heat stencil application to the two- and three-dimensional cases and name them `heat_stencil_2D` and `heat_stencil_3D`. Provide a sequential and an MPI implementation, and use MPI's virtual topologies and derived data types features for the latter. Run your programs with multiple problem and machine sizes.

We provide an example script for submitting jobs. The benchmark was executed with multiple scripts for different configurations.
Attached in the appendix
heat_stencil_2D_seq.cpp
heat_stencil_3D_seq.cpp
heat_stencil_2D_mpi.cpp
heat_stencil_3D_mpi.cpp
heat_stencil_2D_mpi.script

b) How can you verify the correctness of your applications?

In the two 2D case we implemented a function which draws the heat distribution and checked the values by calculating the first two timesteps. In the 3D case we verified the MPI version by comparing it with the sequential version.

Exercise 2:

a) Measure the speedup and efficiency of all three stencil codes for varying problem and machine sizes/mappings. Consider using strong scalability, weak scalability, or both. Justify your choice.

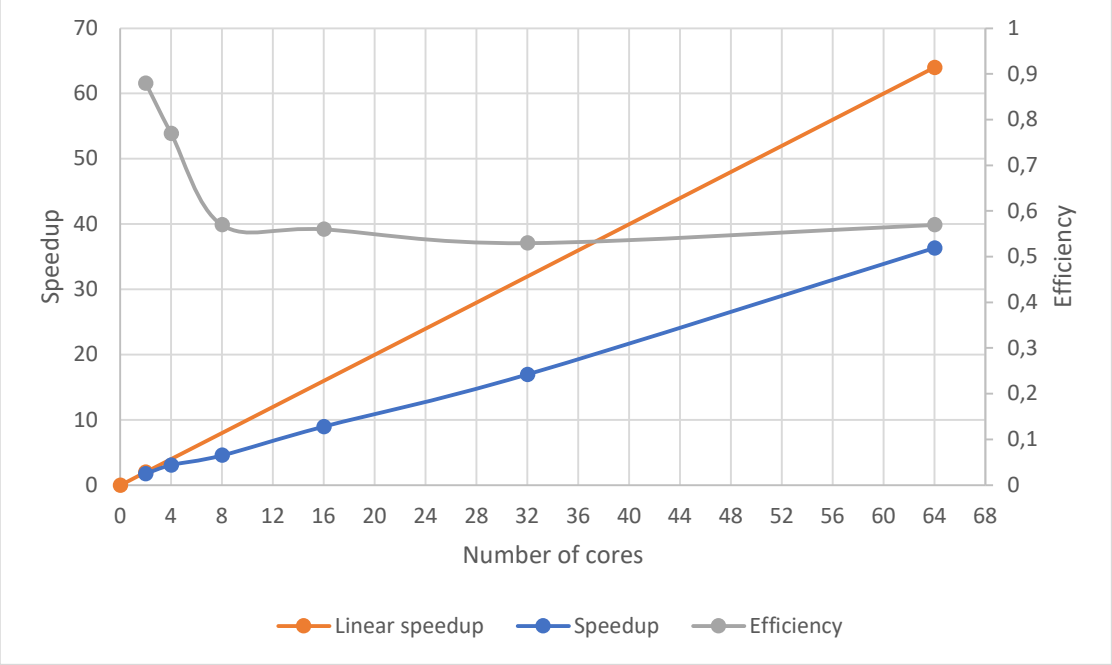
We decided to use strong scaling for all our experiments with the same problem size for all stencil versions. For all three versions we ran two problems sizes to have a better comparison for throughput.
All measurements shown are the median of three executions. In both the sequential and the parallel version, the printing was disabled to omit I/O operation, hence only the actual calculation was measured.
The whole data set can be found in the Appendix.
Assignment03_1D.xlsx
Assignment03_2D.xlsx
Assignment03_3D.xlsx

b) Illustrate the data in appropriate figures and discuss them. What can you observe?

1D Stencil

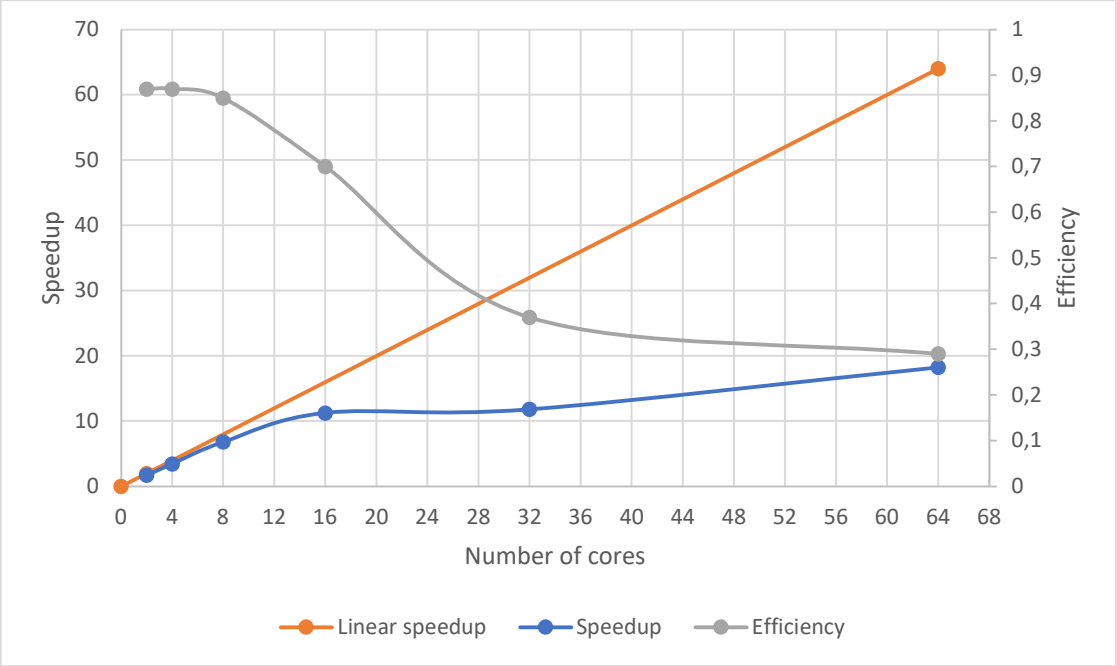
Problem size in each dimension: 16777216

Timesteps: 1000



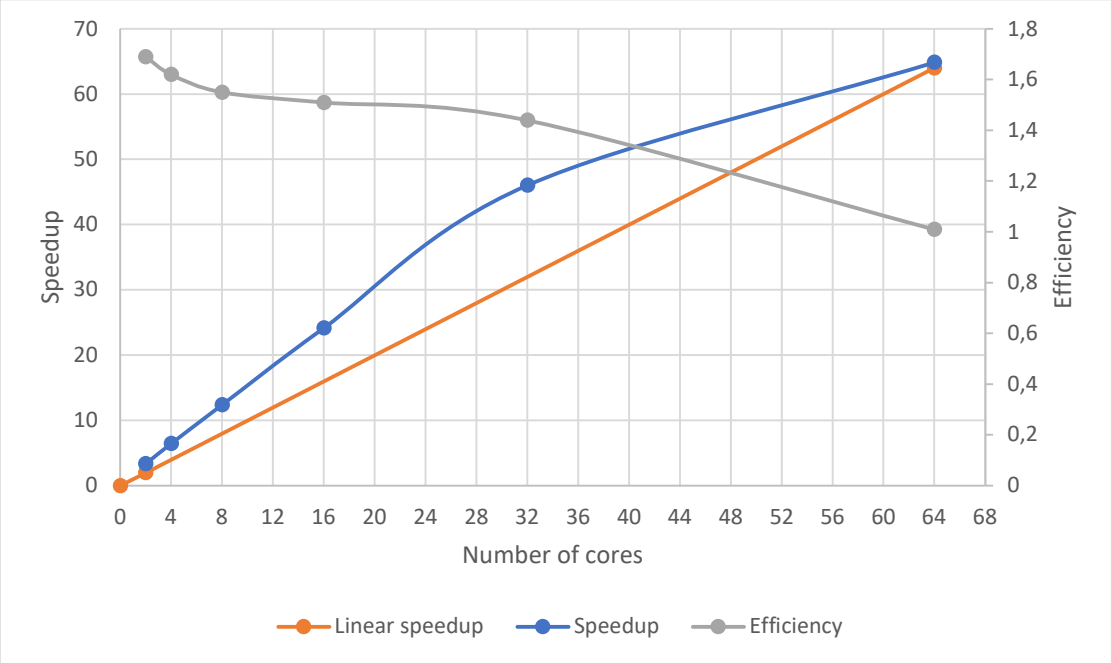
Problem size in each dimension: 262144

Timesteps: 10000

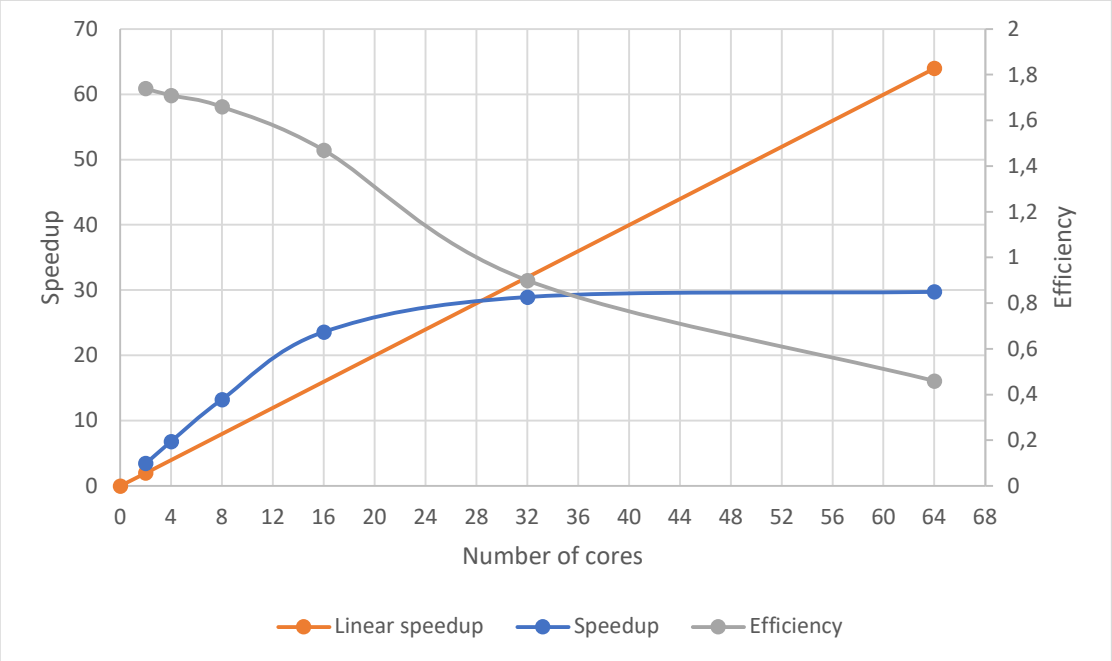


Like the benchmark last week, we can see that with an increasing number of time steps the performance decreases. This is because the communication takes a major part of the whole application.

2D Stencil
Problem size in each dimension: 4096
Timesteps: 1000

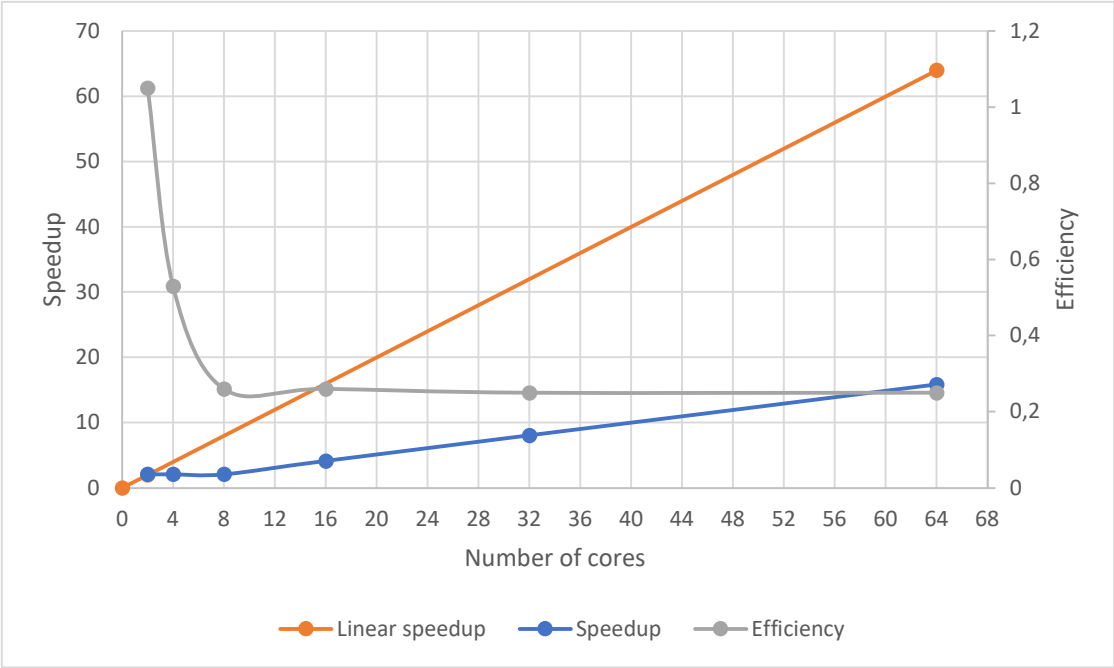


Problem size in each dimension: 512
Timesteps: 10000

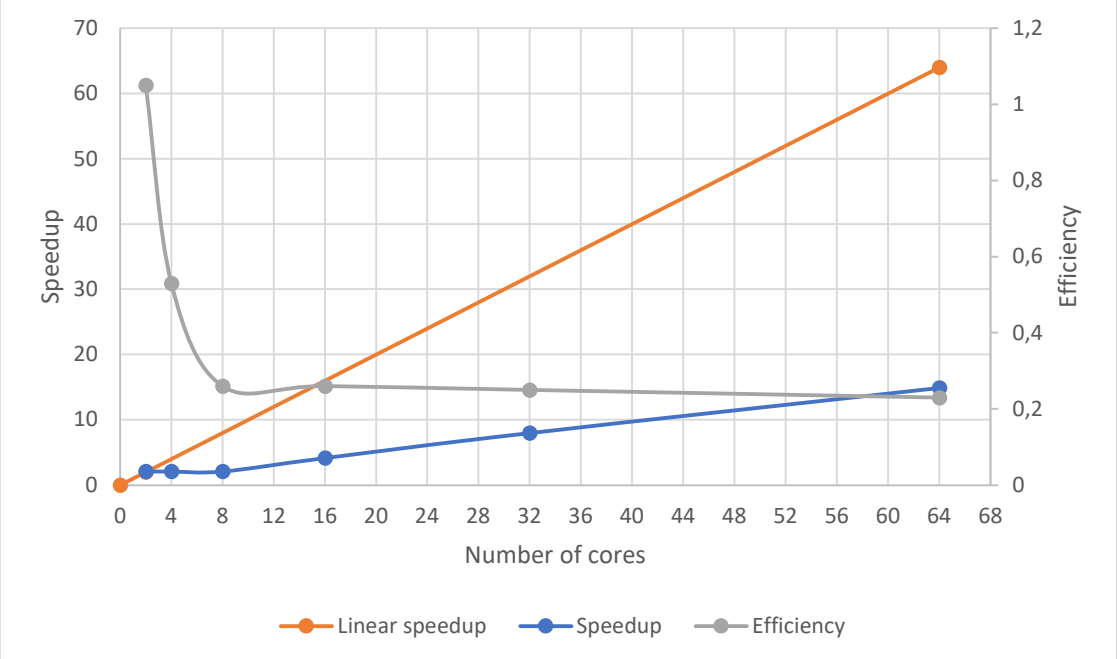


In both cases we see a super linear speedup with its peek at 32 and 16 cores. This might happen because portions of the array fit in a lower level cache now. With a problem size of 4096 in each dimension we achieve efficiencies greater than 1 for every configuration of nodes and cores per node.

3D Stencil
Problem size in each dimension: 256
Timesteps: 1000



Problem size in each dimension: 64
Timesteps: 10000



Independently of the problem size we obtain similar result for both runs. Compared to the other versions, in the 3D case we have the worst speedup and efficiency. Nevertheless, it still shows a linear increase of performance.

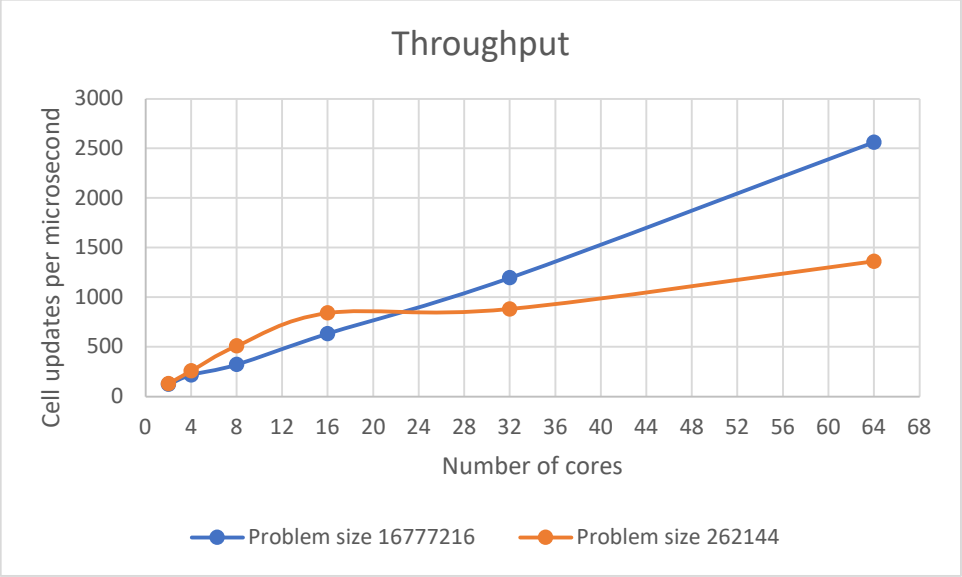
c) Measure and illustrate an application throughput metric. What can you observe?

1D Stencil

Sequential throughput:

Problem size 4096: 70,51

Problem size 512: 74,68

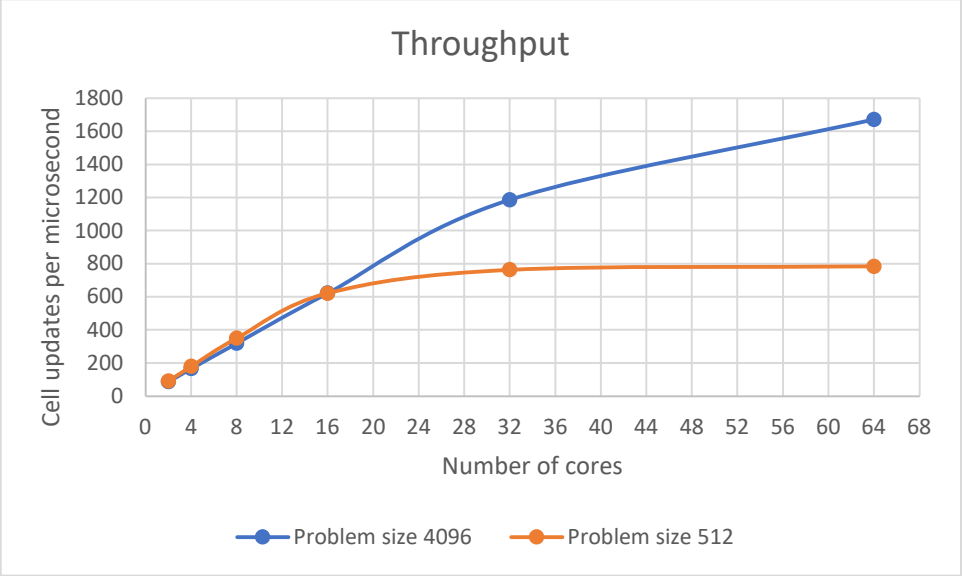


2D Stencil

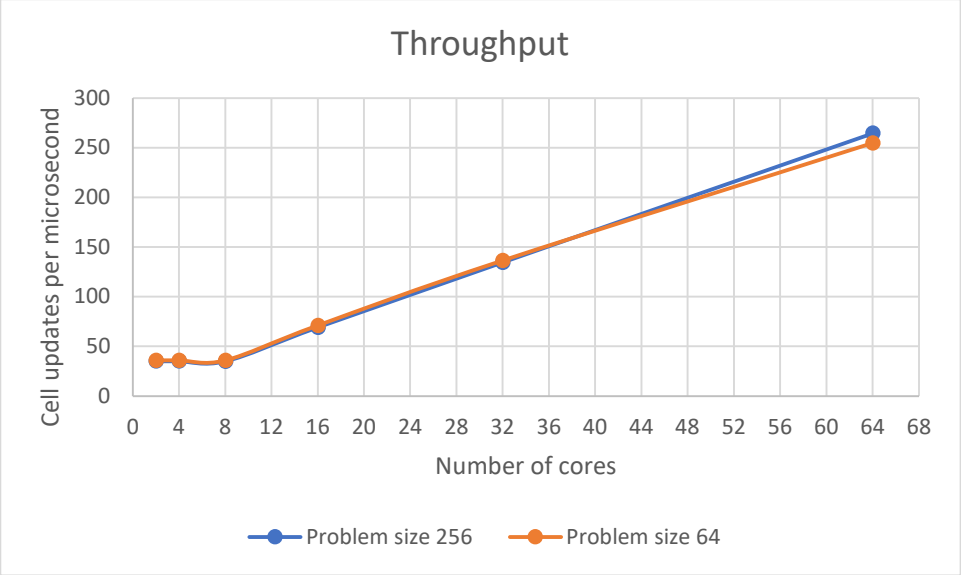
Sequential throughput:

Problem size 4096: 25,78

Problem size 512: 26,39



3D Stencil
Sequential throughput:
Problem size 4096: 16,7
Problem size 512: 17,1



Comparing the Sequential throughput to our parallel versions we obtained very good results for every parallelization. In the 1D and 2D case the runs with less time steps achieved better results. The 3D case shows quite similar throughputs. In all cases we observe a good scalability throughout the benchmarks.

Conclusion:
Overall, the best performance was achieved in the two 2D Stencil. We achieved a super linear speedup because the application can benefit of the use of lower level caches. In general, it can be observed, that the runs with 1000 time steps achieved better performance than the other runs due to the lower amount of communication. Interestingly, when comparing the throughput, in the 2D case the version with 10000 time steps has a better performance than the run with the lower problem size. Again, this might happen because we have a more optimal cache usage with a side length of 4096.