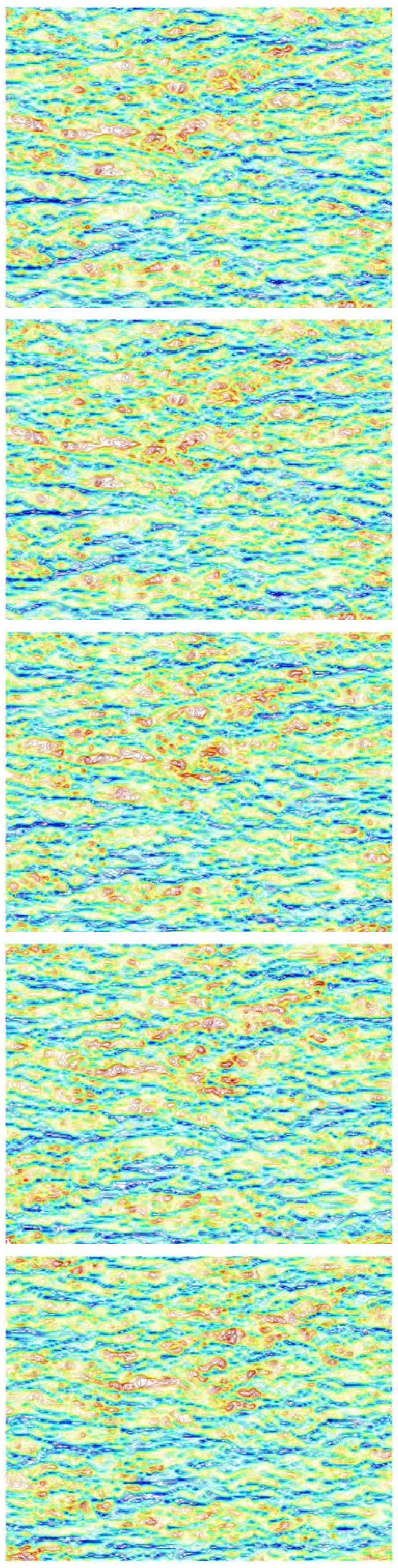
Old-fashioned CPU optimisation of a fluid simulation for investigating turbophoresis

Dr. John Donners (SURFsara, Amsterdam), Prof. Dr. Hans Kuerten (TU/e)
Contact: john.donners@surfsara.nl



Scientific aim

The spectral code performs Direct Numerical Simulation of particle-laden non-isothermal channel flow at a frictional Reynolds number of 950. Both passive particles and inertial particles are tracked in the flow. It is used for three purposes:

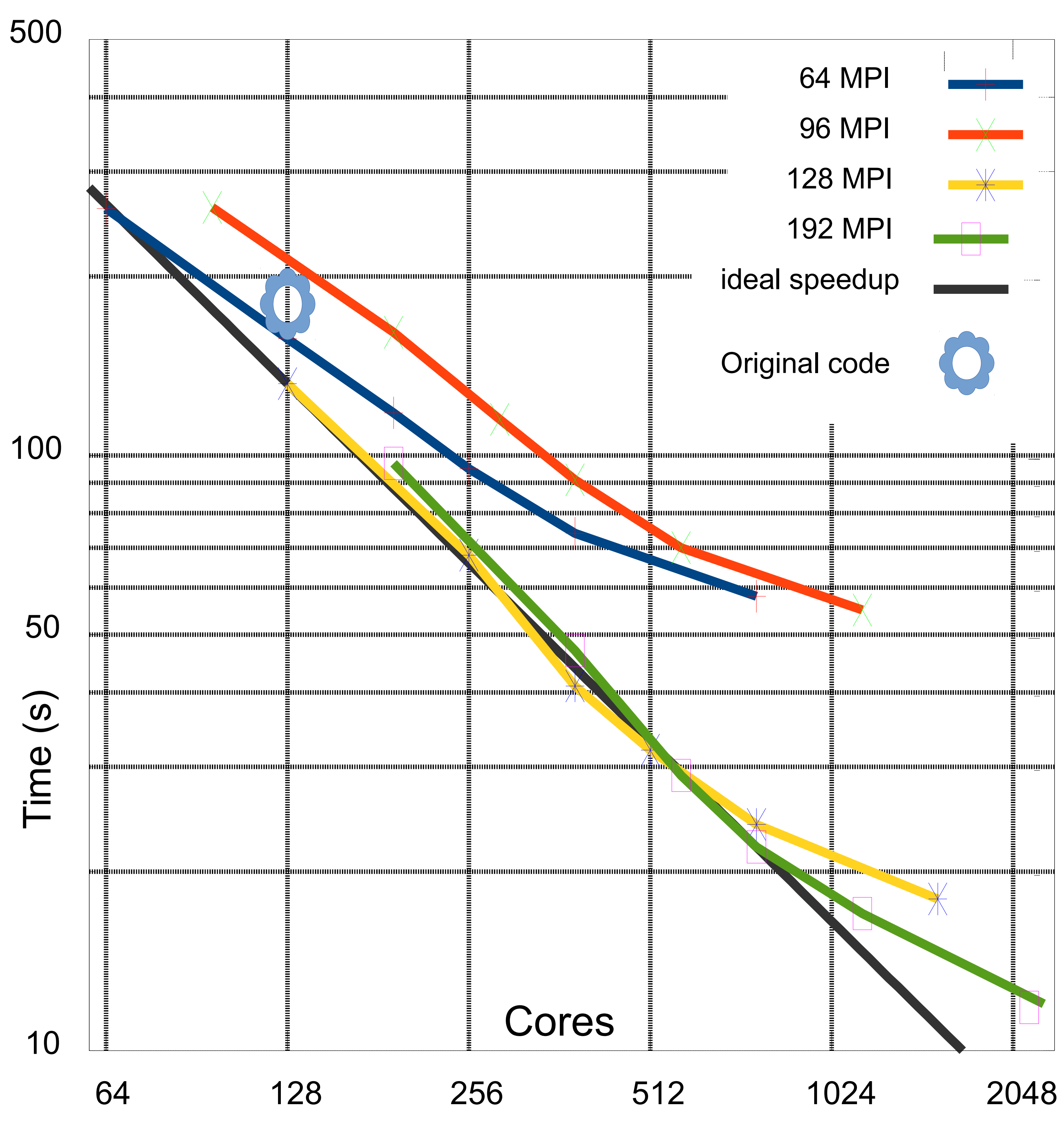
1. Study of turbulent dispersion of passive particles at rather high Reynolds number. This enables
the study of stochastic models for this problem.
2. Study of the validity of eddy diffusivity models for a passive scalar (temperature).
3. Study of the behaviour of inertial particles in inhomogeneous turbulent flow for the development of suitable subgrid models in large-eddy simulation of particle-laden flow.

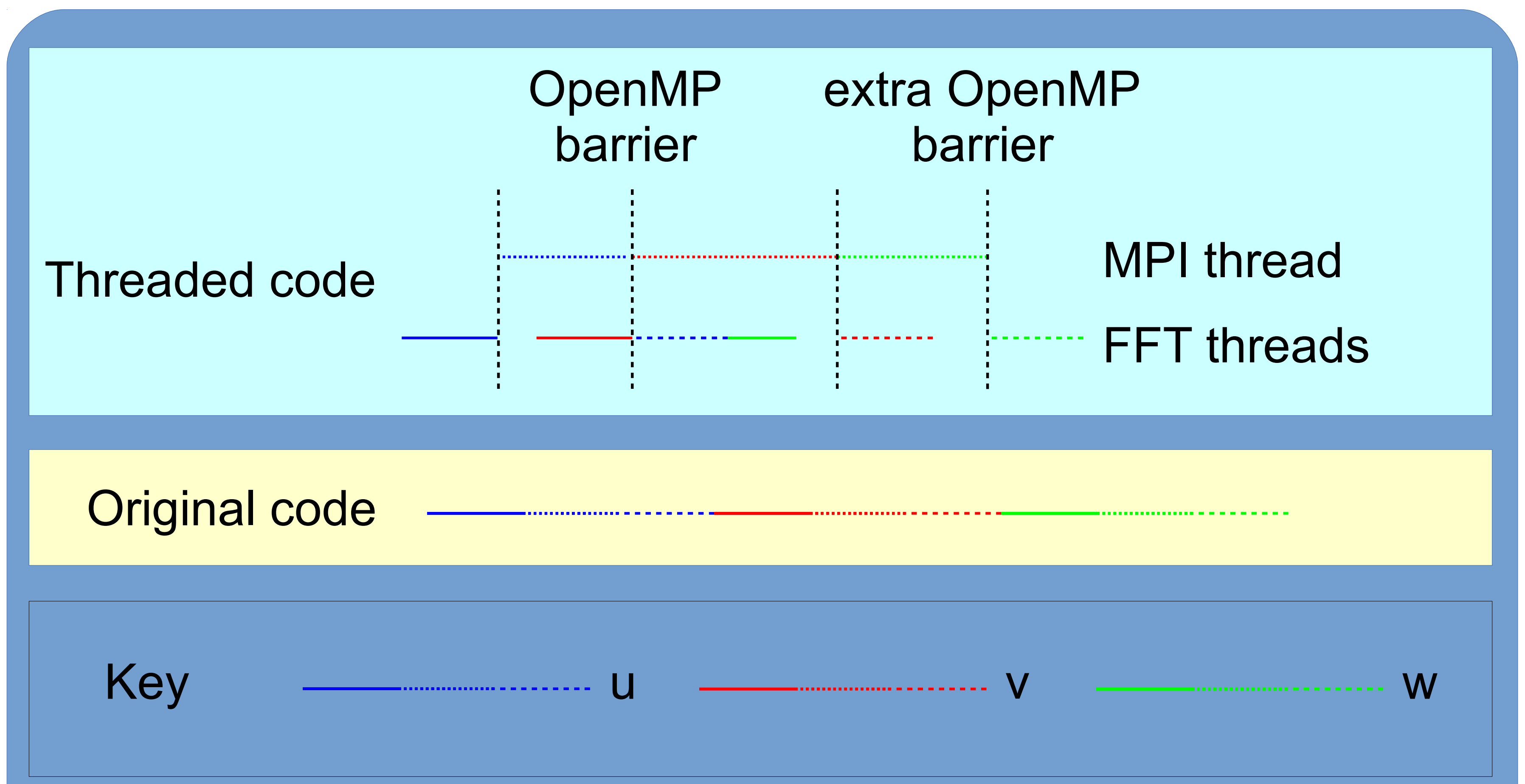
Goal of the project

The resolution is XxYxZ = 768x384x768
The program has been parallelized originally with MPI. The number of MPI tasks is limited to 384 because of the 1D domain decomposition. **The goal is to increase the performance of the model using a threaded OpenMP approach** (strong scaling is used).

Results

Both the MPI communication and the threading performance of the Fourier transforms were optimized using advanced features of the MPI and the FFTW libraries. The OpenMP paradigm has been used to process the MPI communication of one array and the Fourier transforms of another array simultaneously.
Both the MPI communication and the Fourier transforms scale nicely when using more threads, resulting in over 10x speedup.





The figure shows the adopted parallelization strategy. Computations for each variable involve 2 fast Fourier transforms with a communication phase (MPI_Alltoallw) inbetween. One OpenMP thread is reserved for OpenMP, while the other threads are used for FFTW calls.

Discussion: is overlapping MPI and computations a smart idea?

YES! For a multi-threaded code, the use of an **extra core for MPI** can be very advantageous: in this example, **86-98% of the MPI time is overlapped with Fourier transforms.** Ideally, the performance of the code can be doubled with only a modest increase in number of cores used. E.g., when using 5 threads and 1 extra MPI thread, the core increase is only 20%. So, the impact increases with an increasing thread count per process. MPI scales as well, as less processes per node share one network adapter.

NO! In practice, overlap of MPI and computations is severely limited without extra resources. Experiments where **no exta core is reserved for the MPI-thread**, give very poor performance. The kernel is unable to divide the time between MPI and computations efficiently. The same holds for non-blocking collectives as they are implemented in the latest release of MPICH. The use of the `MPI_Test` call to progress the communication is not effective. The performance **gain is about 10%**, if at all.

Is your code too cluttered? And too slow? 2 tips to use HPC libraries more effectively:

Need to calculate many small FFTs? Use the FFTW guru interface!
FFTW can combine many 1D transforms into one call with an excellent multi-threading scalability. Each transform can be strided in memory, without the need for dummy copies.

Preparing dummy arrays for MPI Alltoall? Use MPI derived datatypes instead!
MPI can send and receive non-contiguous blocks of data in one go, with a performance that is often higher than dummy copies. The subroutine `MPI_Type_create_subarray` is a great help to define these derived datatypes.

Reference

Kuerten and Brouwers, Lagrangian statistics of turbulent channel flow at Re$_\tau$ = 950 calculated with direct numerical simulation and Langevin models, **Phys. Fluids 25**, 105108 (2013)