Exercise assignments
Practicalities

Computing servers
We will use CSC’s Cray supercomputer Sisu for the exercises. Log onto Sisu using the provided tnrgXX username and password, e.g.
% ssh -X trng10@sisu.csc.fi
For editing Fortran program files you can use e.g. Emacs editor with or without (the option –nw) X-Windows:
emacs –nw prog.f90
emacs prog.f90
Also other popular editors (vim, nano) are available.

Simple compilation
Compilation and execution are done via the ftn and cc wrapper commands and the aprun scheduler:
% ftn -o my_mpi_exe test.f90
% cc -o my_mpi_exe test.c
% aprun –n 4 ./my_mpi_exe
We will use the default Cray compiling environment. There are also other compilers (GNU and Intel) available on Sisu, which can be changed via (for example)
% module swap PrgEnv-cray PrgEnv-gnu
Use the commands module list and module avail to see the currently loaded and available modules, respectively.

OpenMP programs
Cray compiler processes OpenMP directives by default. In order to disable OpenMP -h noomp flag can be specified. The number of threads used for the execution is specified by environment variable OMP_NUM_THREADS i.e.
aprun –n 1 –d 4 -e OMP_NUM_THREADS=4 ./my_omp_exe

MPI programs
No special options are needed when using wrapper commands ftn and cc. Number of CPU cores to use is specified via the –n flag of aprun, e.g. 4 CPU cores:
aprun –n 4 ./my_mpi_exe
For hybrid programs, one uses both the –n and –d flags:
aprun –n 4 –d 4 -e OMP_NUM_THREADS=4 ./my_hyb_exe
Practicalities

Batch jobs
Larger/longer runs should be submitted via batch system. Example batch job script for hybrid calculation:

```
!/bin/bash
#SBATCH -t 00:10:00
#SBATCH -J hybrid_job
#SBATCH -o out.%j
#SBATCH -e err.%j
#SBATCH -p test
#SBATCH -N 4

export OMP_NUM_THREADS=12

aprun -n 8 -d 12 ./my_hyb_exe
```

The batch script is submitted with `sbatch` command:

```
% sbatch sisu_job.sh
```


Skeleton codes
Skeleton codes both for Fortran 90 and C are provided in the `skeletons` subdirectory. Generally, you do not need to read through all of the code, but look for sections marked with `TODO` for completing the exercises.
MPI & OpenMP recap

1. **Message chain**
   Write a simple program where every processor sends data to the next one. Let `ntasks` be the number of the tasks, and `myid` the rank of the current process. Your program should work as follows:

   - Every task with a rank less than `ntasks-1` sends a message to task `myid+1`. For example, task 0 sends a message to task 1.
   - The message content is an integer array, where each element is initialized to `myid`.
   - The message tag is the receiver’s id number.
   - The sender prints out the number of elements it sends and the tag number.
   - All tasks with rank ≥ 1 receive messages.
   - Each receiver prints out their `myid`, and the first element in the received array.

You can start from the skeleton code `ex1_msg_chain.f90|.c`

2. **Dot product**
   The sekelton code `ex2_dot.f90|.c` implements a simple dot product of two vectors. Parallelize the code using OpenMP.

3. **Parallel heat equation**
   Heat equation is a typical example of partial differential equation in computational physics. Here, we look how initial two dimensional spherical heat distribution evolves in time. The temperature field is presented in uniform two dimensional grid, and at each grid point the temperature in future time step depends on the current values at neighboring grid points. The end of the exercise sheet contains a more thorough explanation of the heat equation for those interested.

   Compile and run a serial reference implementation with the provided Makefiles as
   ```bash
   % make -f Makefile_ex3_F or % make -f Makefile_ex3_C
   % aprun -n 1 ./heat
   ```

   Run the program with e.g. 500x500 grid for 100 steps. Use the `eog` command for visualizing, e.g.
   ```bash
   % eog *.png
   ```
3. Parallel heat equation (cont)

a. Parallelize the serial program with OpenMP by parallelizing the grid update loop. Look for TODOs in `ex3a_heat_omp.f90|.c` (Modify also the Makefile_ex3_F/C as needed).

b. Improve the OpenMP parallelization such that the parallel region is opened and closed only once during the program execution. Now, you need to modify also `ex3b_main.f90|.c`

c. Parallelize the program with MPI, by dividing the grid in column for Fortran or rows for C) and assigning one column/row to one task - a domain decomposition, that is. The tasks are able to update the temperature independently everywhere else than on the column/row boundaries - there the communication of a single column/row with the nearest neighbor is needed (the board is aperiodic, so the first and last columns/rows have only a single neighbor). This is realized by having additional ghost layers on each of the local columns/rows, that contain the boundary data of the neighboring tasks. When printing out the board, all tasks send their local parts to one task that prints out the temperature field. Insert the proper MPI routines into a skeleton codes `ex3c_main.f90|.c` and `ex3c_heat_mpi.f90|.c` (search for “TODO”s).
4. Hybrid “Hello world”

Write a simple hybrid program where each OpenMP thread communicates using MPI. Implement a test case where the threads of task 0 send their thread id to corresponding threads in other tasks (see the picture). Remember to initialize the MPI library using MPI_Init_thread routine and use the MPI_THREAD_MULTIPLE thread affinity mode. See `ex4_hybridhello` for skeleton.

5. Hybrid version of heat equation

Parallelize the heat program with hybrid MPI+OpenMP paradigm, using three different approaches:

a. Fine-grained version, where the halo exchange is performed outside the parallel region (this basically means just inserting the OpenMP-parallelized update loop into the MPI program). Solution in `ex5a_heat_hyb`.

b. Version, where the threads are alive throughout the execution but only the master thread performs communication. Solution in `ex5b_heat_hyb`.

c. Version employing multiple-thread MPI communication (threads communicating directly their counterparts in another MPI task). Solution in `ex5c_heat_hyb`.
6. Something’s broken?

By carrying out performance analysis, find out the reasons why the provided version of the heat code (`ex3c_main_mpi / ex3c_heat_mpi`) does not scale. Indeed it should scale, as it is a simple domain decomposition with thin halos! Alternatively, you can carry out the eight-step procedure discussed in the lecture for your own application.
Advanced MPI exercises

7. Vector datatype
Write a Fortran (or C) program that sends a row (a column for C) of a matrix from one process to another by using your own datatype. A skeleton code is provided in `ex7_vector_type.c|.f90`.

8. Subarray datatype
Write a program that sends a block of a matrix from one process to another by using your own datatype. A skeleton code is provided in `ex8_subarray_type.c|.f90`.

9. Two dimensional heat equation
The skeleton code `ex9_heat_2d.c|.f90` implements a two dimensional decomposition utilizing cartesian process topology for the halo exchange. Utilize user defined datatypes in the halo exchange during the time evolution. MPI contains also a contiguous datatype `MPI_Type_contiguous` which can be used in order to use user defined datatypes both in x- and y-directions in the halo exchange.

Utilize user defined datatypes also in the I/O related communication.

10. Simple one-sided communication
Write a simple program where set of processes exchange data with one-sided communication. Each process “sends” its rank and “receives” the rank of previous process (rank-1). Employ periodic boundary conditions (i.e. process ntasks-1 “sends” to process 0 etc.)

a) Use only PUT operation
b) Use only GET operation

11. One-sided communication in the heat
Rewrite the halo exchange of column/row boundaries in the heat program using one-sided communication.
Parallel I/O exercises

12. Trying it out

Take 8 MPI processes and initialize a vector of (say) 100 elements in each of them to their MPI ranks. Then dump this data to disk such that the full vector (800 elements) is written to

a) A single file by one process (spokesman strategy)
b) Into two files by two processes (subset of writers)
c) Into a single file with MPI I/O
d) Into two files with MPI I/O

Use binary format all the way. A solution is in `ex12_pario.c|.f90`.

13. Checkpointing with MPI I/O

Add a feature to the heat equation solver program that enables one to start the program from a completed situation (i.e. not from scratch every time). This checkpointing will dump the situation of the simulation to disk (e.g.) after every few tens of iterations; in a form that can be read in afterwards. If a checkpoint file is present, the program will replace the initial state with the one in the restart file. Use MPI I/O to accomplish this. A starting point (in skeletons/) and a solution (in answers/) are provided in `ex13_heat.c|.f90`
14. Matrix multiplication

Use the BLAS \texttt{dgemm} routine for performing the matrix-matrix product in \texttt{ex14\_matmul(.c|f90)}. Investigate the performance of naïve implementation and the library call with different matrix sizes. Have a look at http://www.netlib.org for a reference to \texttt{dgemm}.

15. Dense linear algebra 1

Find a LAPACK routine for solving the linear system $\textbf{A}\textbf{x}=\textbf{b}$ for the matrices provided in \texttt{ex15\_lapack1(.c|f90)}. Have a look at http://www.netlib.org for a reference to LAPACK routines.

16. Dense linear algebra 2

Find a LAPACK routine for solving the eigenproblem $\textbf{A}\textbf{x}=\lambda \textbf{x}$ for the matrices provided in \texttt{ex16\_lapack2(.c|f90)}. Have a look at http://www.netlib.org for a reference to LAPACK routines.
Appendix: Heat equation

Heat equation is a partial differential equation that describes how a temperature field in a given region varies over time. In the most general form the equation can be written as:

\[
\frac{\partial u}{\partial t} = \alpha \nabla^2 u
\]

where \( u \) is the temperature depending on the spatial coordinates and time, \( u(x, y, z, t) \) and \( \alpha \) is thermal diffusivity constant, a given material parameter. For numerical solution, the temperature field can be discretized on a grid, and the Laplacian can be evaluated with the finite-difference method which in two dimension has the form:

\[
\nabla^2 u(i, j) = \frac{u(i - 1, j) - 2u(i, j) + u(i + 1, j)}{(\Delta x)^2} + \frac{u(i, j - 1) - 2u(i, j) + u(i, j + 1)}{(\Delta y)^2}
\]

The time evaluation can be treated with explicit method, where the new temperature \( u^{m+1} \) after a time-step \( \Delta t \) is given by

\[
u^{m+1}(i, j) = u^m(i, j) + \Delta t \alpha \nabla^2 u^m(i, j)
\]

The full time variation is determined now by the initial condition \( u(t=0) = u^0 \) and the boundary conditions.

Note: the explicit time evolution is table only when

\[
\Delta t < \frac{1}{2 \alpha \frac{(\Delta x \Delta y)^2}{(\Delta x)^2 + (\Delta y)^2}}
\]