EXERCISE ASSIGNMENTS

Practicalities

Compilation and running OpenMP programs

Simple example program test.f90 that can be used to test the compiler:

```
program test
  implicit none
!$OMP PARALLEL
  print *, ‘hello world!’
!$OMP END PARALLEL
end program test
```

Compilation can be done using the gcc and gfortran compilers:

```
    % gfortran -fopenmp test.f90 -o test
    % OMP_NUM_THREADS=4 ./test
    Hello world!
    Hello world!
    Hello world!
    Hello world!
```

Advanced MPI I exercises

1. Vector datatype

Write a Fortran (or C) program that sends a row (a column for C) of a matrix from one process to another. A skeleton code is provided in exercises/ex_adv_mpi1_vector(.c|.f90).

a) use manual packing/unpacking and predefined MPI datatypes
b) define your own datatype.

2. Subarray datatype

Write a program that sends a block of a matrix from one process to another. A skeleton code is provided in exercises/ex_adv_mpi2_subarray(.c|.f90).

a) use manual packing/unpacking and predefined MPI datatypes
b) define your own datatype.

3. Two dimensional heat equation

The file exercises/ex_adv_mpi3_heat(.c|.f90) contains a parallel code for two dimensional heat equation implemented using predefined MPI datatypes. Modify program so that user defined datatypes are used both in the halo exchange during the time evolution as well as in the I/O related communication. MPI contains also a contiguous datatype MPI_Type_contiguous which can be used in order to utilize user defined datatypes both in x- and y-directions in the halo exchange.

Advanced MPI II exercises

1. Simple write and read

Write numbers 0-99 to a file in parallel using MPI I/O. Verify the write by reading the file with different number of processes than in writing. A skeleton code is provided in exercises/ex_adv_mpi2_1_simple_io(.c|.f90). Try to do multiple writes in the same process and investigate how the file pointer is updated when using MPI_File_write. Hint: MPI provides a function MPI_File_get_position.

2. Using file views I

Write numbers 0-99 to a file. Distribute the numbers to the processes in strides, (e.g. 0, 4, 8, ... for rank 0 with 4 processes) but have the numbers in the correct order in the file. Verify the write by reading. A skeleton is provided in `exercises/ex_adv_mpi2_2_fileview(.c|.f90)`.

3. Using file views II

Write an array that is ditributed in two dimension to a file using MPI I/O A skeleton is provided in `exercises/ex_adv_mpi2_3_fileview2(.c|.f90)`

4. Restart in heat equation

Implement to the heat equation code in `exercises/ex_adv_mpi2_4_heat(.c|.f90)` a feature which dumps the data to a file by certain intervals and allows restarting from a saved information. Utilize MPI I/O and have all the MPI tasks to participate in I/O.

OpenMP exercises

1. Parallel region and data clauses

Take a look at the exercise skeleton `exercises/ex_omp1_variables.c` (or `ex_omp1_variables.f90`). Add an OpenMP parallel region around the block where the variables `Var1` and `Var2` are printed and manipulated. What results do you get when you define the variables as shared, private or firstprivate? Explain why do you get different results.

2. Work sharing of a simple loop

The file `exercises/ex_omp2_sum(.c|.f90)` implements a skeleton for the simple summation of two vectors **C=A+B**. Add the computation loop and add the parallel region with work sharing directives so that the vector addition is executed in parallel.

3. Dot product and race condition

The file `exercises/ex_omp3_dotprod(.c|.f90)` implements a simple dot product of two vectors. Try to parallelize the code by using **omp parallel** or **omp for** pragmas. Are you able to get same results with different number of threads and in different runs? Explain why the program does not work correctly in parallel. What is needed for correct computation?

4. Reduction and critical

Continue with the previous dot product example and use **`reduction`** clause to compute the sum correctly. Implement also an alternative version where each thread computes its own part to a private variable and then use a **`critical`** section after the loop to compute the global sum. Try to compile and run your code also without OpenMP. Do you get exactly same results in all cases?

5. Using the OpenMP library functions

Write a simple program that uses **`omp_get_num_threads`** and **`omp_get_thread_num`** library functions and prints out the total number of active threads as well as the id of each thread.

6. One dimensional Jacobi solver

The skeleton code `exercises/ex_omp6_jacobi(.c|.f90)` contains a working serial code for solving one dimensional Poisson equation using the Jacobi method. The code contains already some OpenMP pragmas. Investigate the code and try if if works correctly also in parallel. Note that compiling the C version requires the math library, that is, you have to add **`-lm`** linker flag during compilation of the program. Insert appropriate execution control clauses (**`barrier`**, **`single`**, etc) so that the parallel version works correctly.

OpenMP exercises

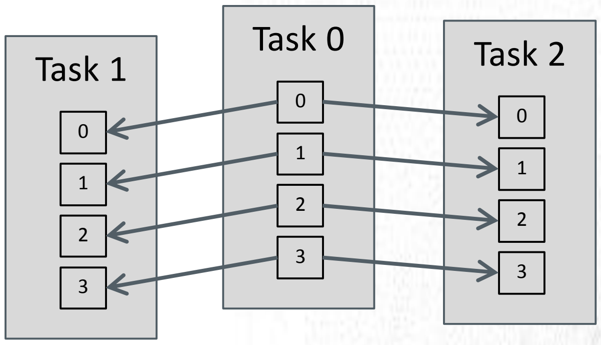
Bonus exercise 1: Two dimensional heat equation

The skeleton code `exercises/ex_omp_B1_heat(.c|.f90)` contains a working serial code for two dimensional heat equation. Parallelizae the code using OpenMP along the instructions given in the comments in the code. Try to minimize the need for thread synchronization, that is, try to maximize the size of the parellel region. Run the parallelized version with different number of threads and check the obtained speed-up. Remember to compare the obtained results with serial code.

Hybrid programming exercises

1. 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. Take a look at the exercise skeleton `exercises/ex_hyb1_hello.c` (or `ex_hyb1_hello.f90`).



2. Hybrid version of the heat equation

Parallelize the heat equation program `exercises/ex_hyb2_heat(.c|.f90)` with hybrid MPI+OpenMP paradigm, starting from the MPI implementation and 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).

b) Version, where the threads are alive throughout the execution but only the master thread performs communication.

c) Version employing multiple-thread MPI communication (threads communicating directly their counterparts in another MPI task). Remember to set the variable MPICH_MAX_THREAD_SAFETY to multiple.

HPC library exercises

1. Matrix multiplication

Use the BLAS **dgemm** routine for performing the matrix-matrix product in exercises/ex_lib_1_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 **dgemm**.

2. Dense linear algebra I

Find a LAPACK routine for solving the linear system **Ax=b** for the matrices provided in exercises/ex_lib_2_lapack1(.c|.f90). Have a look at http://www.netlib.org for a reference to LAPACK routines.

3. Dense lienar algebra II

Find a LAPACK routine for solving the eigenproblem **Ax=λx** for the matrices provided in exercises/ex_lib_3_lapack2(.c|.f90). Have a look at http://www.netlib.org for a reference to LAPACK routines).