

Prepare directories

You will find the file *mpi.tar*. Expand the file with the command:

```
tar xvf mpi.tar
```

A directory **mpi** is created with subdirectories for the individual exercises

Compilation

You compile MPI programs written in C with

```
mpicc -O3 -o hello hello.c
```

Program execution

```
mpirun -np <#processes> <executable> <arguments>
```

Time measurements

Use the MPI function

```
double MPI_Wtime();
```

to measure wall clock execution time. It returns the time in seconds.

Intel Traceanalyzer

Intel's traceanalyzer is a powerful trace visualization tool for MPI programs.

- Prepare your environment for tracing:
 - `module load mpi_tracing`
- Compile your program with

```
mpicc -O3 -vtrace -o shift shift.c
```
- Run the program
 - `mpirun -np 5 shifts 8`
- The program will generate a file *shifts.stf*
- Analyze the trace file
 - `traceanalyzer shift.stf`

.More Information about Intel traceanalyzer:

<http://www.lrz-muenchen.de/services/software/parallel/vampir/>

Exercise 1) Hello World

Parallelize the HelloWorld program. Adapt the output so that it writes

```
HELLO WORLD. This is process <process rank>.
```

Run the program with different numbers of threads.

Exercise 2) Shift

1. Compile and run the parallel implementation of the shift program from the lecture. Pass the number of shift operations via an argument at program start. Analyze the program with the Intel traceanalyzer.
2. Change the implementation such that *MPI_Sendrecv* is used for the communication in the shift steps.

Exercise 3) Parallelize the computation of PI

Parallelize the sequential program *pi.c*. It computes an approximation of PI. Compute the speedup for large *n*.

Exercise 4) Master Slave

Parallelize the sequential program *loadbalance.c*. Function *icalman* in *loadbalance.c* is called from a parallel loop. The execution time of the function varies depending on the arguments.

Transform the program such that:

- Process 0 acts as master and all the other processes as slaves.
- The master executes the entire program. But, instead of computing *icalman* by itself, the master sends the parameters to a waiting slave. The slave computes the function and returns the function value.
- Analyze the parallel implementation with VAMPIR.

Exercise 5) Shift (Continue)

3. Change the generation of the output. Allocate a full size array in process 0. Use *MPI_Gather* or *MPI_Gatherv* to collect the values of the distributed array in process 0. Process 0 then writes the array to stdout.
4. Change the communication to nonblocking communication. Execute *MPI_Isend* and *MPI_Irecv* first and overlap local computation with communication.

Exercise 6) IO

Adapt your Hello World program such that:

1. Each process writes its string to an own file.
2. The processes write their strings into a common file in ascending order.

Exercise 7) Parallelize Euler Method

The program *euler.c* uses the Forward Euler method.

1. Parallelize the program with *MPI_Send* and *MPI_Recv*.
2. Analyze the communication with traceanalyzer
3. Change the implementation and use nonblocking communication, i.e. *MPI_Isend* and *MPI_Irecv*.
4. Analyze the communication with traceanalyzer.