This course material has been developed from the SimDataLab “Enginee-ring in Energy and Mobility” (Project NHR@KIT) and the Engineering Competence Center (Project s5 of bwHPC) – both being parts of the Steinbuch Centre for Computing (SCC) at the Karlsruhe Institute of Technology

Tutorial: BwUniCluster 2.0/HoreKa
Starting Parallel Computations with OpenFOAM

In this tutorial we will learn how to submit a job (using a script-file) for execution of parallel computation on the BwUniCluster 2.0 or HoreKa. Parallel simulations can be used to speed up the process of simulation by a significant margin. However, they have to be executed according to the guidelines below in order to ensure that the simulation runs correctly. Also, special preprocessing is necessary for OpenFOAM in case of parallel simulations.

First of all, it is important to understand what the logical structure of our clusters looks like.

1. Logical structure of the supercomputers

The bwUniCluster 2.0 or HoreKa are parallel computers which are divided into logical units called nodes. Each of those nodes consists of multiple cores with memory attached to them (each core can be thought as a single PC-processor). All components of the whole logical structure can communicate with each other via a communication protocol called MPI, which allows for parallel computing. MPI (Message Passing Interface) is necessary for the data exchange between the processors – both between the cores within a node or between the cores of multiple nodes, see the Figure.

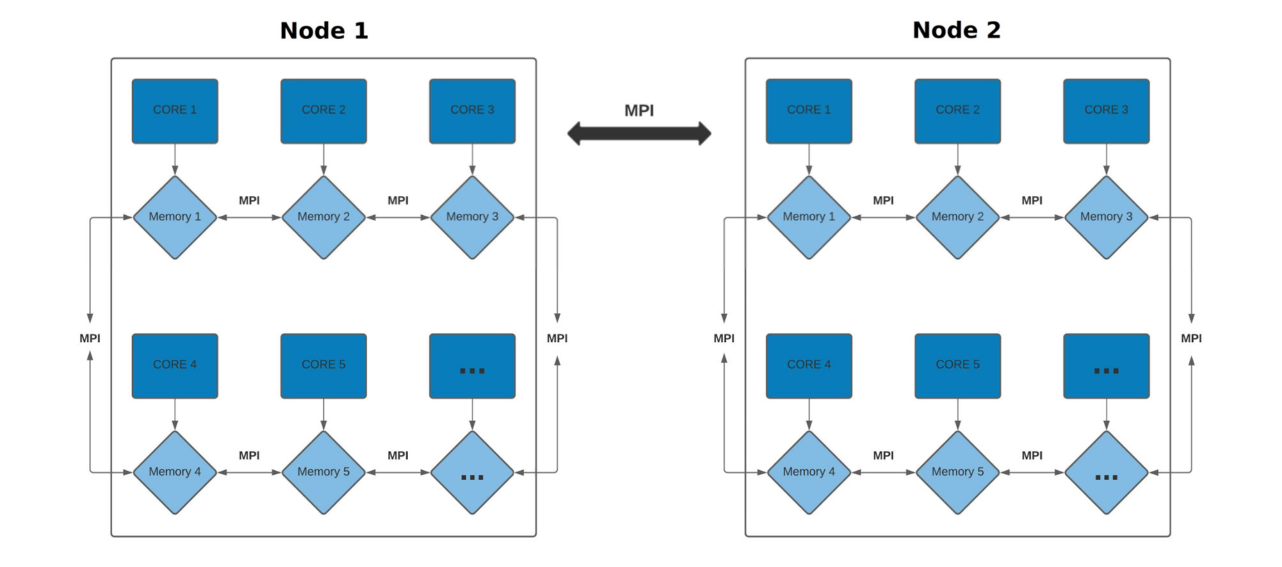

Fig. 1 The logical structure of the cluster

When submitting a job, the user should specify the number of cores and the number of nodes for the particular simulation. There are different queues for testing, for using only one node, or using two or more nodes. The job submission is carried out by one single command followed by the name of a file that describes the desired number of cores, or nodes, the memory, the job-queue and the name of the OpenFOAM-solver. But before submitting the job, a special preprocessing of OpenFOAM needs to be done.

2. Preprocessing of OpenFOAM for parallel computations

First, we need to decide on how many parallel cores we would like to start the simulation. For starters, it is always a good idea to take a small number of cores – in our case four. We are going to prepare OpenFOAM for this exact number of cores.

Log onto the server and go to your workspace. Then go to the directory where you wish to start OpenFOAM, i.e. – to the directory with the case you will simulate.

In the 'system' subdirectory you can (usually) find the decomposeParDict file where you can set up the way subdomains are created; if this file is not available, you can take it from an appropriate tutorial case. Notice the number of subdomains which is equal to the product of the coordinates of the partition vector (in this example: for the (1 4 1) partition this number is equal to 1x4x1 = 4). This value will be needed in the creation of the job file later in the tutorial.

A typical decomposeParDict file usually looks something like this:

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 4;  ←――――  the mesh will be divided into 4 subdomains
method  simple;  ←――――  for this method we also specify the number of splits in each direction
coeffs;  ←――――  some methods (like "scotch") do not require coefficients
{
   n        (1 4 1);  ←――――――――  4 subdomains (splits) are created in the y-direction
}

//*************************************************************************//
```

After you have configured the file to your liking execute the following commands in your working directory:

```
$>   blockMesh
$>   setFields
$>   decomposePar
```

3. Creating the job-file

After that, log onto the cluster in your command-line interpreter and go to your working directory in your workspace. Create an empty job file using the following command:

```
$>   touch {name of the the job-file}.sh
```

(For example: *$>   touch job1_development_queue_2_cores.sh*)

Open the job-file in order to edit it. A typical job file takes the following form:

//

```
#!/bin/bash  ←————————————————  The header of the file

#SBATCH --partition dev_multiple  ←———  the job will be submitted to the queue dev_multiple
#SBATCH –nodes=2  ←———————————  two nodes will be used
#SBATCH --ntasks-per-node=2  ←———  the number of cores to be used on each node
#SBATCH –time=00:30:00  ←———————  the job will run for the maximum of 30 minutes
#SBATCH –mem=8000mb  ←—————————  the job may use the max. of 8 Gb
#SBATCH –job-name=coursejob  ←————  the name of the job given by the user

module purge                           ┐
module load cae/openfoam/v2106-impi    ├—   loading OPENFoam on the execute core
source $FOAM_INIT                      ┘

compressibleInterFoam  ←———————————  the name of the solver
```

//

Note!: On HoreKa use dev_cpuonly instead of dev_multiple.

There is also an alternative way of creating the job file – you can go to:

https://wiki.bwhpc.de/e/BwUniCluster_2.0_Slurm_common_Features

where the user can copy a job file and modify it.

4. Submitting the job (commands of the SLURM job scheduler)

After editing and saving the job file we can submit the job using the following command from our working directory:

$> sbatch {name of the job-file}.sh

(For example: *$> sbatch job1_development_queue_4_cores.sh*)

which sends our job on the development queue for single nodes and then causes all of the commands in the job-file to be executed step by step. In order to see the list of all your pending jobs you can use the command:

$> squeue --start

We can also see a very detailed information about the state of our job using the following command:

$> scontrol show job

After our job has been completed, we can use the following OpenFOAM command to combine results from all subdomains into one:

```
$>   reconstructPar
```

The results of our simulation can be downloaded from our working directory onto our own PC where we can then perform the visualization in Paraview on our own PC. This is the recommended way for visualization of small and medium sized grids.