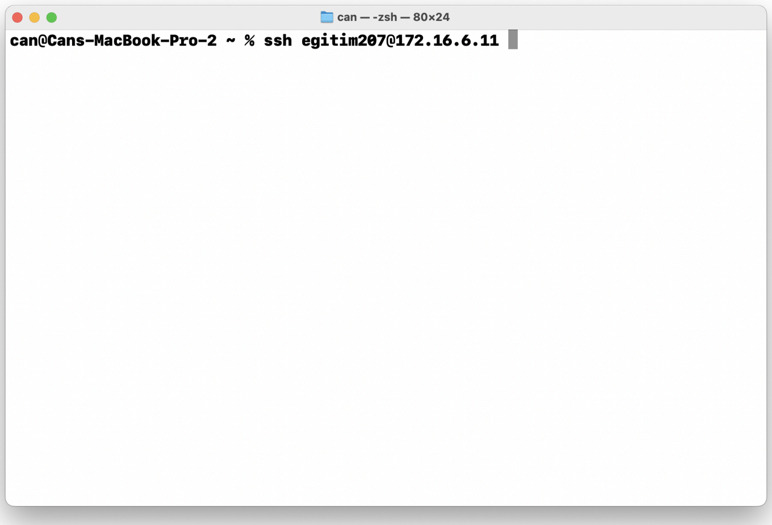
BASIC ORFOZ USAGE

Connection

After installing and activating the necessary VPN, you can establish an ssh connection from the terminal with your username and host address as follows. The host address for the ARF machine containing the ORFOZ queue is 172.16.6.11.



```
can@Cans-MacBook-Pro-2 ~ % ssh egitim207@172.16.6.11
```

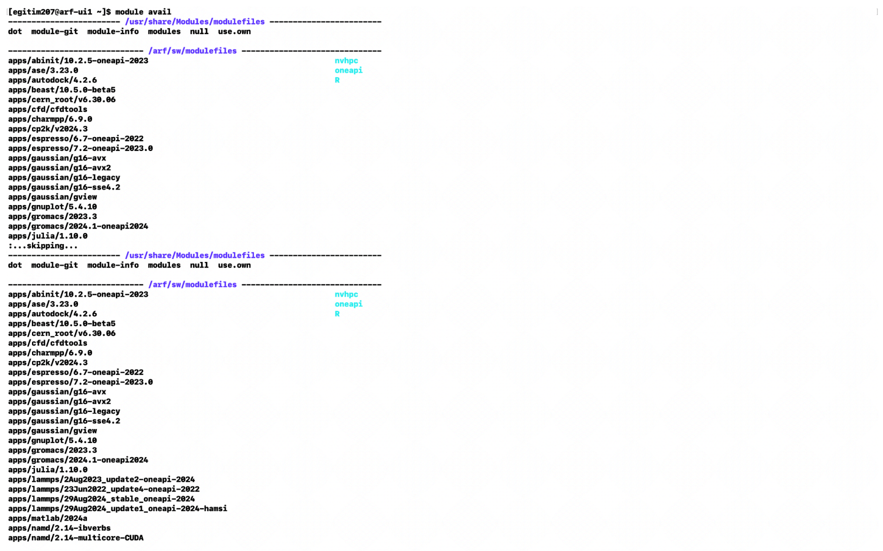
Connection using VSCode

You can also connect to the system using VSCode's remote ssh extension. A better interface option for those who don't want to work from the terminal. I recommend using this option, especially if you are not using a UNIX-based operating system (like Windows).

After the connection is established, it will direct you to your main directory (/arf/home/username). You will do your work in this directory. You will write and compile your code here. **However, do not run your code here, otherwise your account will be suspended.** You will run your code on compute clusters as described in the rest of the document.

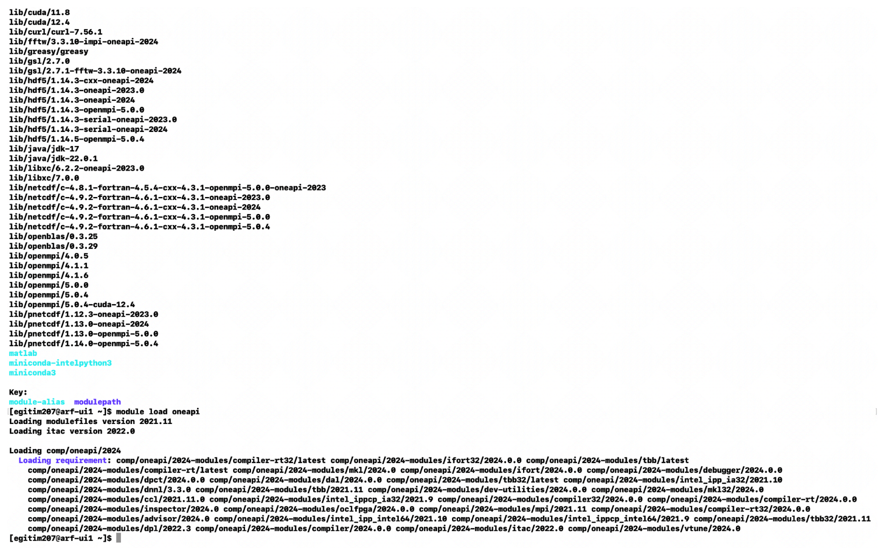
Module Load

The libraries required to compile and run your code are already installed on the ARF machine. However, you must load the ones you need. You can see all the libraries/frameworks installed on the system using the "module avail" command.



```
[egitim207@arf-ui1 ~]$ module avail
------------------------ /usr/share/Modules/modulefiles ------------------------
dot  module-git  module-info  modules  null  use.own

----------------------------- /arf/sw/modulefiles -----------------------------
apps/abinit/10.2.5-oneapi-2023                                      nvhpc
apps/ase/3.23.0                                                     oneapi
apps/autodock/4.2.6                                                 R
apps/beast/10.5.0-beta5
apps/cern_root/v6.30.06
apps/cfd/cfdtools
apps/charmpp/6.9.0
apps/cp2k/v2024.3
apps/espresso/6.7-oneapi-2022
apps/espresso/7.2-oneapi-2023.0
apps/gaussian/g16-avx
apps/gaussian/g16-avx2
apps/gaussian/g16-legacy
apps/gaussian/g16-sse4.2
apps/gaussian/gview
apps/gnuplot/5.4.10
apps/gromacs/2023.3
apps/gromacs/2024.1-oneapi2024
apps/julia/1.10.0
:...skipping...
------------------------ /usr/share/Modules/modulefiles ------------------------
dot  module-git  module-info  modules  null  use.own

----------------------------- /arf/sw/modulefiles -----------------------------
apps/abinit/10.2.5-oneapi-2023                                      nvhpc
apps/ase/3.23.0                                                     oneapi
apps/autodock/4.2.6                                                 R
apps/beast/10.5.0-beta5
apps/cern_root/v6.30.06
apps/cfd/cfdtools
apps/charmpp/6.9.0
apps/cp2k/v2024.3
apps/espresso/6.7-oneapi-2022
apps/espresso/7.2-oneapi-2023.0
apps/gaussian/g16-avx
apps/gaussian/g16-avx2
apps/gaussian/g16-legacy
apps/gaussian/g16-sse4.2
apps/gaussian/gview
apps/gnuplot/5.4.10
apps/gromacs/2023.3
apps/gromacs/2024.1-oneapi2024
apps/julia/1.10.0
apps/lammps/2Aug2023_update2-oneapi-2024
apps/lammps/23Jun2022_update4-oneapi-2022
apps/lammps/29Aug2024_stable_oneapi-2024
apps/lammps/29Aug2024_update1_oneapi-2024-hamsi
apps/matlab/2024a
apps/namd/2.14-ibverbs
apps/namd/2.14-multicore-CUDA
```

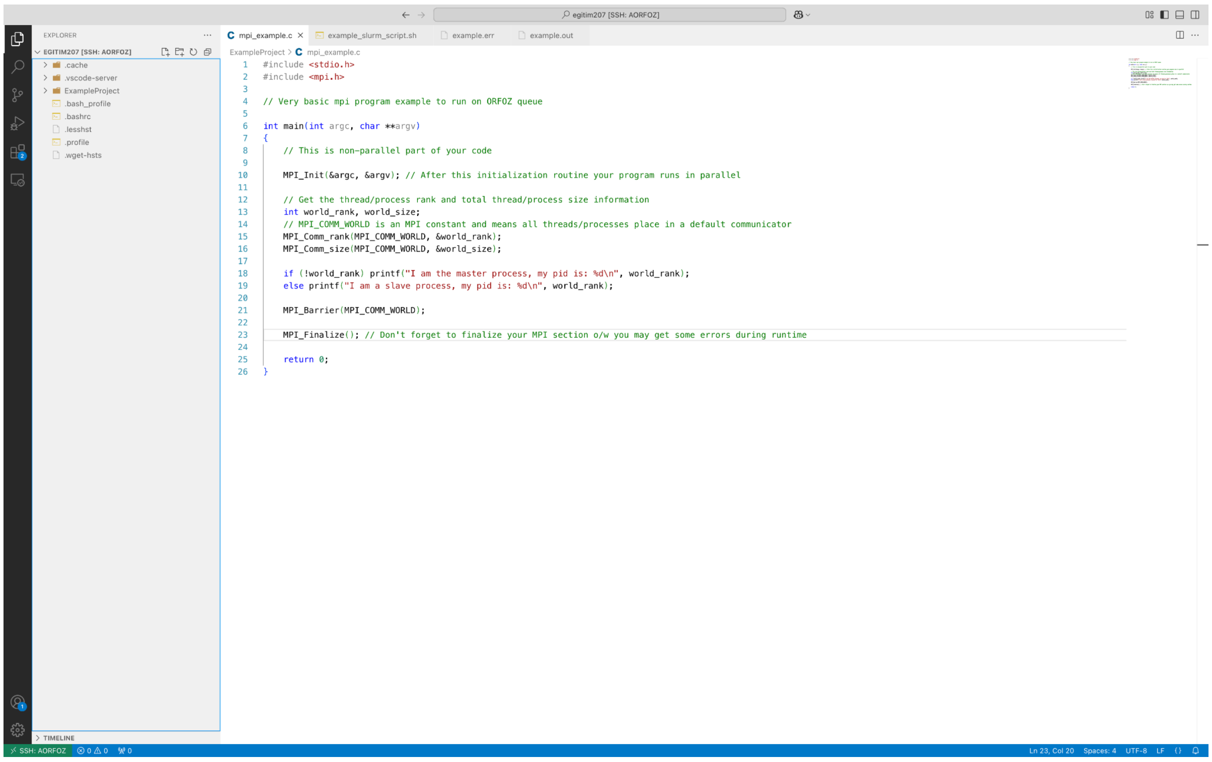
At this stage, the only library/module you need is MPI. As you can see, there are multiple MPI distributions. (OpenMPI, MPICH, impi, oneAPI) I recommend that you use Intel MPI provided by oneAPI or the latest OpenMPI version in this system. You can load the desired library by giving the directory where it is installed with the "module load" command (module load oneapi or module load lib/openmpi/5.0.4). With these modules, you will have both the gcc-based mpicc compiler that you can compile your code with and the MPI runtime environment.



```
lib/cuda/11.8
lib/cuda/12.4
lib/curl/curl-7.56.1
lib/fftw/3.3.10-impi-oneapi-2024
lib/greasy/greasy
lib/gsl/2.7.0
lib/gsl/2.7.1-fftw-3.3.10-oneapi-2024
lib/hdf5/1.14.3-cxx-oneapi-2024
lib/hdf5/1.14.3-oneapi-2023.0
lib/hdf5/1.14.3-oneapi-2024
lib/hdf5/1.14.3-openmpi-5.0.0
lib/hdf5/1.14.3-serial-oneapi-2023.0
lib/hdf5/1.14.3-serial-oneapi-2024
lib/hdf5/1.14.5-openmpi-5.0.4
lib/java/jdk-17
lib/java/jdk-22.0.1
lib/libxc/6.2.2-oneapi-2023.0
lib/libxc/7.0.0
lib/netcdf/c-4.8.1-fortran-4.5.4-cxx-4.3.1-openmpi-5.0.0-oneapi-2023
lib/netcdf/c-4.9.2-fortran-4.6.1-cxx-4.3.1-oneapi-2023.0
lib/netcdf/c-4.9.2-fortran-4.6.1-cxx-4.3.1-oneapi-2024
lib/netcdf/c-4.9.2-fortran-4.6.1-cxx-4.3.1-openmpi-5.0.0
lib/netcdf/c-4.9.2-fortran-4.6.1-cxx-4.3.1-openmpi-5.0.4
lib/openblas/0.3.25
lib/openblas/0.3.29
lib/openmpi/4.0.5
lib/openmpi/4.1.1
lib/openmpi/4.1.6
lib/openmpi/5.0.0
lib/openmpi/5.0.4
lib/openmpi/5.0.4-cuda-12.4
lib/pnetcdf/1.12.3-oneapi-2023.0
lib/pnetcdf/1.13.0-oneapi-2024
lib/pnetcdf/1.13.0-openmpi-5.0.0
lib/pnetcdf/1.14.0-openmpi-5.0.4
matlab
miniconda-intelpython3
miniconda3

Key:
module-alias  modulepath
[egitim207@arf-ui1 ~]$ module load oneapi
Loading modulefiles version 2021.11
Loading itac version 2022.0

Loading comp/oneapi/2024
  Loading requirement: comp/oneapi/2024-modules/compiler-rt32/latest comp/oneapi/2024-modules/ifort32/2024.0.0 comp/oneapi/2024-modules/tbb/latest
    comp/oneapi/2024-modules/compiler-rt/latest comp/oneapi/2024-modules/mkl/2024.0 comp/oneapi/2024-modules/ifort/2024.0.0 comp/oneapi/2024-modules/debugger/2024.0.0
    comp/oneapi/2024-modules/dpct/2024.0.0 comp/oneapi/2024-modules/dal/2024.0.0 comp/oneapi/2024-modules/tbb32/latest comp/oneapi/2024-modules/intel_ipp_ia32/2021.10
    comp/oneapi/2024-modules/dnnl/3.3.0 comp/oneapi/2024-modules/tbb/2021.11 comp/oneapi/2024-modules/dev-utilities/2024.0.0 comp/oneapi/2024-modules/mkl32/2024.0
    comp/oneapi/2024-modules/ccl/2021.11.0 comp/oneapi/2024-modules/intel_ippcp_ia32/2021.9 comp/oneapi/2024-modules/compiler32/2024.0.0 comp/oneapi/2024-modules/compiler-rt/2024.0.0
    comp/oneapi/2024-modules/inspector/2024.0 comp/oneapi/2024-modules/oclfpga/2024.0.0 comp/oneapi/2024-modules/mpi/2021.11 comp/oneapi/2024-modules/compiler-rt32/2024.0.0
    comp/oneapi/2024-modules/advisor/2024.0 comp/oneapi/2024-modules/intel_ipp_intel64/2021.10 comp/oneapi/2024-modules/intel_ippcp_intel64/2021.9 comp/oneapi/2024-modules/tbb32/2021.11
    comp/oneapi/2024-modules/dpl/2022.3 comp/oneapi/2024-modules/compiler/2024.0.0 comp/oneapi/2024-modules/itac/2022.0 comp/oneapi/2024-modules/vtune/2024.0
[egitim207@arf-ui1 ~]$ 
```

Example Code, Compile and Run

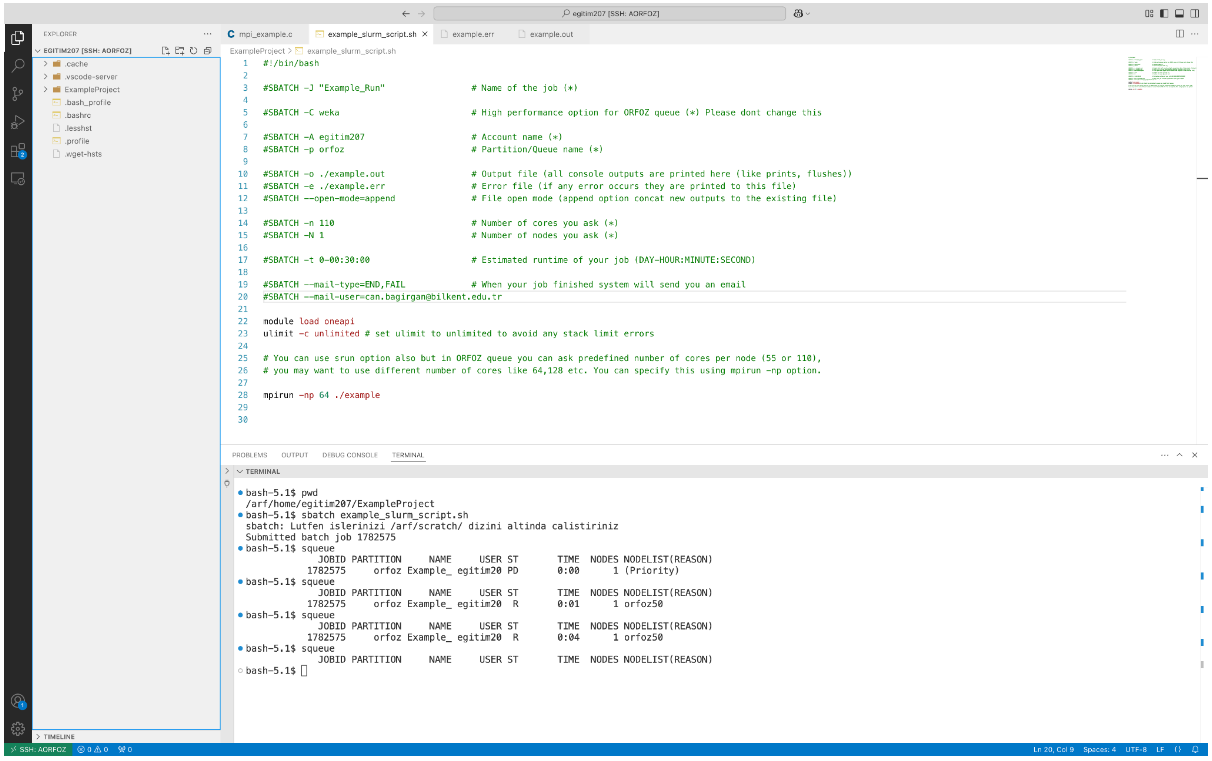
Below you can see a very simple MPI code. Each process prints an output to the console according to its rank. We will simply see how to compile this code, then run it and get the output.



```c
#include <stdio.h>
#include <mpi.h>

// Very basic mpi program example to run on ORFOZ queue

int main(int argc, char **argv)
{
    // This is non-parallel part of your code

    MPI_Init(&argc, &argv); // After this initialization routine your program runs in parallel

    // Get the thread/process rank and total thread/process size information
    int world_rank, world_size;
    // MPI_COMM_WORLD is an MPI constant and means all threads/processes place in a default communicator
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    if (!world_rank) printf("I am the master process, my pid is: %d\n", world_rank);
    else printf("I am a slave process, my pid is: %d\n", world_rank);

    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Finalize(); // Don't forget to finalize your MPI section o/w you may get some errors during runtime

    return 0;
}
```

First of all, you can compile your code in your own directory, there is no problem with that. However, do not forget to use the MPI compiler (mpicc). (Please do not forget to prepare a Makefile, even if it is simple, as requested in the assignment.)

You need to use the SLURM scheduler to run your code. There is more than one way to do this, I will show you how to run a SLURM script. Below you can see an example script.



```bash
#!/bin/bash

#SBATCH -J "Example_Run"                # Name of the job (*)

#SBATCH -C weka                         # High performance option for ORFOZ queue (*) Please dont change this

#SBATCH -A egitim207                    # Account name (*)
#SBATCH -p orfoz                        # Partition/Queue name (*)

#SBATCH -o ./example.out                # Output file (all console outputs are printed here (like prints, flushes))
#SBATCH -e ./example.err                # Error file (if any error occurs they are printed to this file)
#SBATCH --open-mode=append              # File open mode (append option concat new outputs to the existing file)

#SBATCH -n 110                          # Number of cores you ask (*)
#SBATCH -N 1                            # Number of nodes you ask (*)

#SBATCH -t 0-00:30:00                   # Estimated runtime of your job (DAY-HOUR:MINUTE:SECOND)

#SBATCH --mail-type=END,FAIL            # When your job finished system will send you an email
#SBATCH --mail-user=can.bagirgan@bilkent.edu.tr

module load oneapi
ulimit -c unlimited # set ulimit to unlimited to avoid any stack limit errors

# You can use srun option also but in ORFOZ queue you can ask predefined number of cores per node (55 or 110),
# you may want to use different number of cores like 64,128 etc. You can specify this using mpirun -np option.

mpirun -np 64 ./example


```

```
bash-5.1$ pwd
/arf/home/egitim207/ExampleProject
bash-5.1$ sbatch example_slurm_script.sh
sbatch: Lutfen islerinizi /arf/scratch/ dizini altinda calistiriniz
Submitted batch job 1782575
bash-5.1$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1782575     orfoz Example_ egitim20 PD       0:00      1 (Priority)
bash-5.1$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1782575     orfoz Example_ egitim20  R       0:01      1 orfoz50
bash-5.1$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1782575     orfoz Example_ egitim20  R       0:04      1 orfoz50
bash-5.1$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
bash-5.1$ 
```

Here are some SLURM options/configurations that start with #SBATCH. Some of these (the ones I marked with *) are mandatory. Let's go through them in order.

-J : İşinizin ismi. İstediğiniz herhangi bir şey olabilir ancak sistem tarafından işinizin bu isimle görünür olacağını unutmayın.

-C weka: This is a mandatory feature for ORFOZ. It is used to allocate one core per socket for system and background work and to increase the performance of the other cores. Please do not change this.

-A : You need to write your account name here.

-p : Partition/queue name. You only have permission to use the ORFOZ queue on this machine, so type orfoz here.

-o, -e : Options where you specify the file you want to print the output to after your code is run and the file you want possible errors to be written to. (Everything you press on the console is written directly to this file) If you do not specify a separate error file with the -e option, errors will also be printed to the output file.

-N, -n : These are the most important. There are over 500 nodes in ORFOZ. A node has 2 sockets and each socket has 56 cores. In other words, there are 112 cores in a node. With -N, you specify how many nodes you want from the system. With -n, you specify how many cores you want in total. However, as a limitation of ORFOZ, you can only request 55 or 110 cores from a node (due to weka, one core from each socket belongs to the system). For example, with -n 110 -N 1, you can get 1 node and 110 cores. Or you can request 110 cores with -n 110 -N 2. However, if you do not enter a number that is 55 or multiples, it will give an error.

-t : An estimate of how long your job will take. If you enter a time that is too long, SLURM may make you wait in line longer because the job will take longer. However, if you enter a time that is too short, your job will be cut off as soon as the time runs out. I recommend entering a reasonable amount.

-mail : With the mail options, it sends you an e-mail when your work is done. However, this system does not work very well in ORFOZ, so you may not use it :)

After the SBATCH options, we are loading the oneapi that we loaded for compile here because this job will run on another node. I also recommend that you maximize the stack size to be used on the node with ulimit.

After preparing all these, we are ready to run the code. We write our run string here in the same way we normally run it. If you run it with mpirun, you can use as many tasks as you want with -np. You can eliminate the 55 or 110 core restriction this way. However, let's say you allocated 55 cores but used only 4 of them in the code with -np 4. In this case, ORFOZ will give you an efficiency warning after a while and suspend your account. So run your code with at least 32 cores (prepare your input a little larger accordingly). You can switch to more core numbers later (like 64, 128, 256). You can run your code with srun instead of mpirun, but you will have to use 55 and its multiples.

Finally, you must enter the command "sbatch your_scriptname.sh" from the terminal to give your job to SLURM.

When you submit your project, you must have this SLURM script in your zip file.

Some Useful Commands
sbatch: To give/run your jobs.
squeue: To see the status of your running/waiting jobs.
module list: To see the modules that you have loaded.
module load: To load modules.
module avail: To see the modules that the system has.
scontrol show partition=orfoz: To see the current status of ORFOZ queue.

Notes and Recommendations

- This guide is for ORFOZ queue and use this queue as much as possible. BARBUN and HAMSI are not second options but backup queues for us.This guide is for ORFOZ queue and use this queue as much as possible. BARBUN and HAMSI are not second options but backup queues for us.
- You can also run your serial code here. However, to avoid efficiency errors, instead of taking 1 core with -np 1, take all cores (55) and only one core works serially while giving the others a free job (a small loop, but not too big so that it doesn't finish after the serial job).
- ORFOZ is not a development system but a production system. So it is not recommended to develop your code here, you should only bring your ready code and run it with high core numbers here.
- You can simply use "scp" to upload any file from your local to ORFOZ, and vice versa.

If you have any questions about the system other than these, please ask me. (can.bagirgan@bilkent.edu.tr)