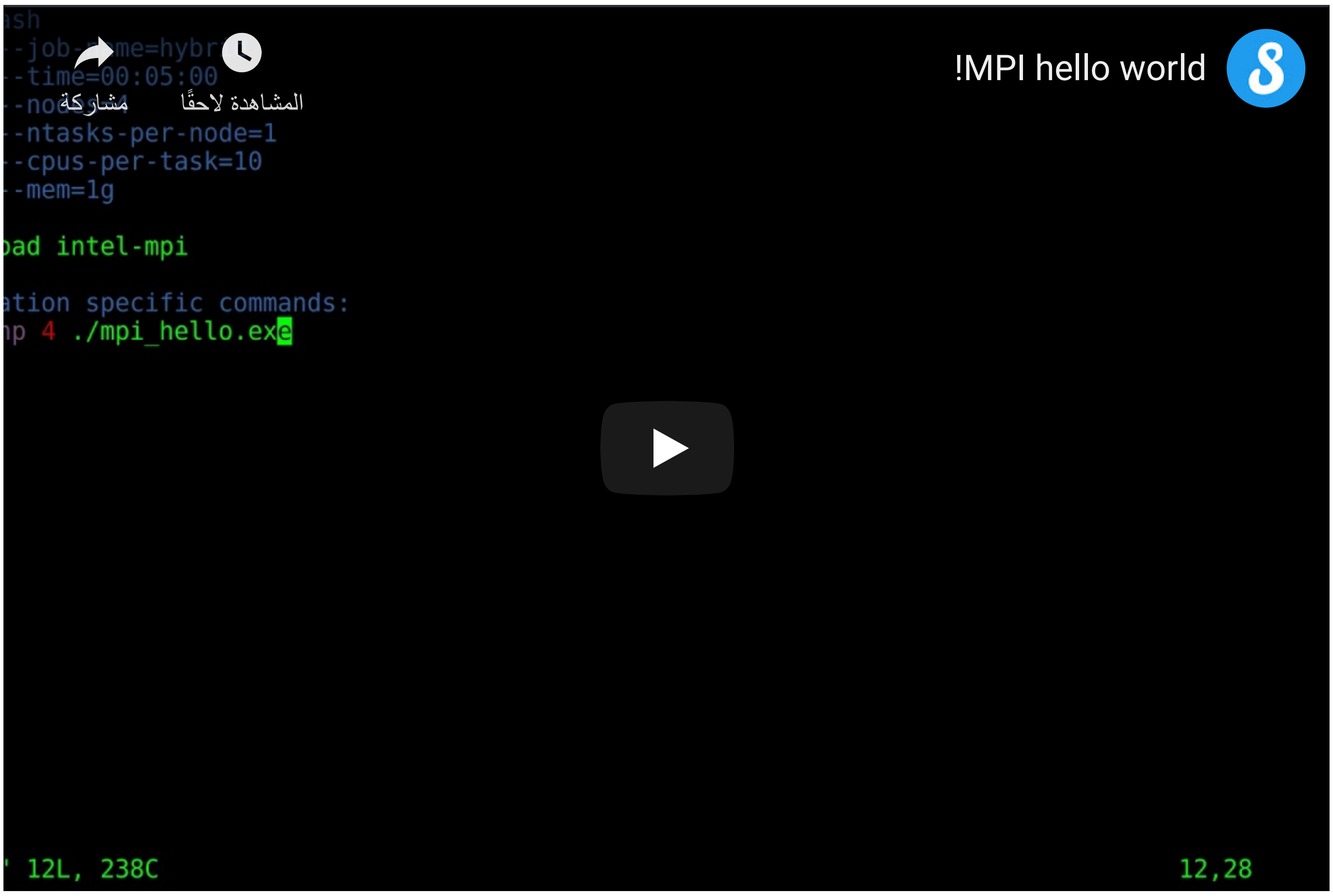
MPI - hello world!

In this lesson, we will show you a basic **MPI hello world** code and also discuss how to run an MPI program. The lesson will cover the basics of initializing MPI and running an MPI job across several processes.



```
ash
--job-name=hybr
--time=00:05:00
--nodes=4
--ntasks-per-node=1
--cpus-per-task=10
--mem=1g

oad intel-mpi

ation specific commands:
np 4 ./mpi_hello.exe



' 12L, 238C                                                    12,28
```

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;

    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d"
           " out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Let’s get introduced with the functions and environment veriables used in the code:

- `MPI_Init`, all of MPI’s global and internal variables are constructed.
- `MPI_Comm_size` returns the size of a communicator and the built-in `MPI_COMM_WORLD` encloses all of the processes in the job, so this call should return the amount of processes that were requested for the job.
- `MPI_Comm_rank` returns the rank of a process in a communicator. Each process inside of a communicator is assigned an incremental rank starting from zero.
- `MPI_Finalize` is used to clean up the MPI environment. No more MPI calls can be made after this one.

Running the MPI hello world: Let’s create a `makefile` to compile the above code. Note that the `makefile` looks for the `MPICC` environment variable and the `mpicc` program in your installation is really just a wrapper around `gcc`, and it makes compiling and linking all of the necessary MPI routines much easier!

```
EXECS=mpi_hello_world
MPICC?=mpicc

all: ${EXECS}

mpi_hello_world: mpi_hello_world.c
    ${MPICC} -o mpi_hello_world mpi_hello_world.c

clean:
    rm ${EXECS}
```

You can now be able to compile running the command `make` in the same folder where the source and makefile reside. Now you need to create a `host_file` with the name of the desired nodeslist where you want to run the code, for example:

```
>>> cat machine.file
node1
node3
node6
node10
```

Finally, you are done! now you should be able to run the compiled code on the above hosts, by running the command:

```
mpirun -np 4 -machinefile machine.file  mpi_hello_world
```

You may want to `export` this for the environment variable `MPI_HOSTS` as follows:

```
export MPI_HOSTS=machine.file
```