Installing OpenMPI

```
apt-get install libopenmpi-dev openmpi-bin openmpi-doc openmpi-common
```

Compiling and running MPI programs

```
mpicc test.c
```

Starts 4 processes. Possibly on different machines.
They are identical but have different ids.
Works with non-MPI programs.

```
mpirun –np 4 a.out
mpirun –np 3 date
```

```
./a.out
```

Works but starts only **one** process.

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Start MPI Process

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Get the id (rank)

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Get the total number of processed

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

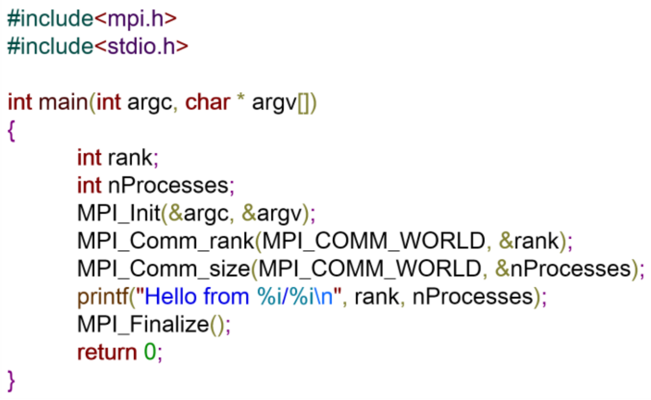
Print hello from all processes.

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

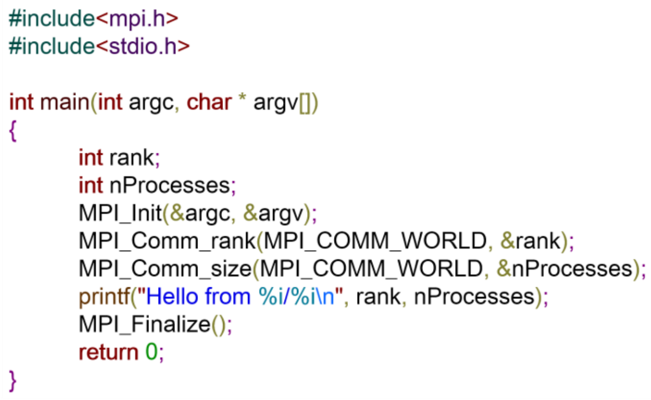
Stop the MPI environment.

MPI example executed



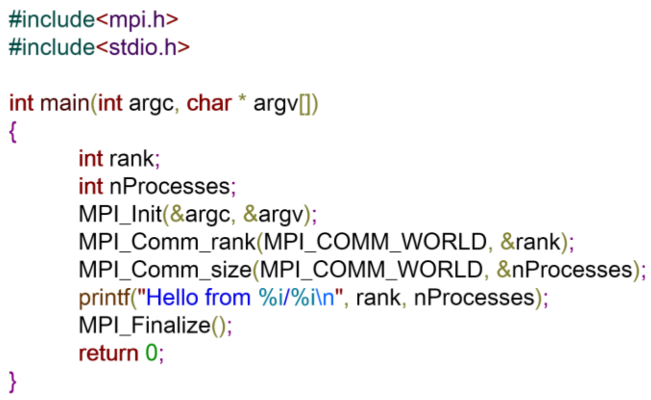
```
#include<mpi.h>
#include<stdio.h>

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Hello from 0/4



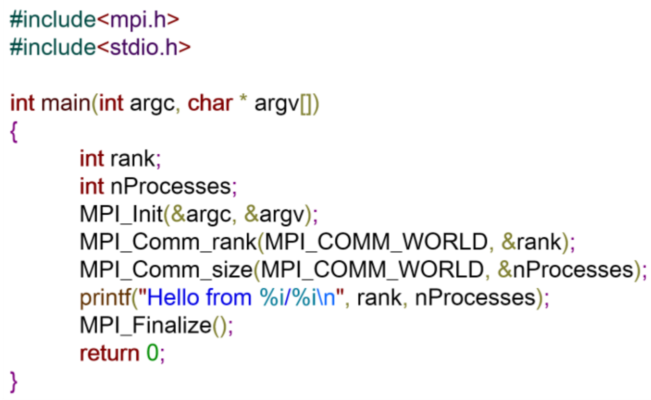
```
#include<mpi.h>
#include<stdio.h>

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Hello from 3/4



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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Hello from 2/4



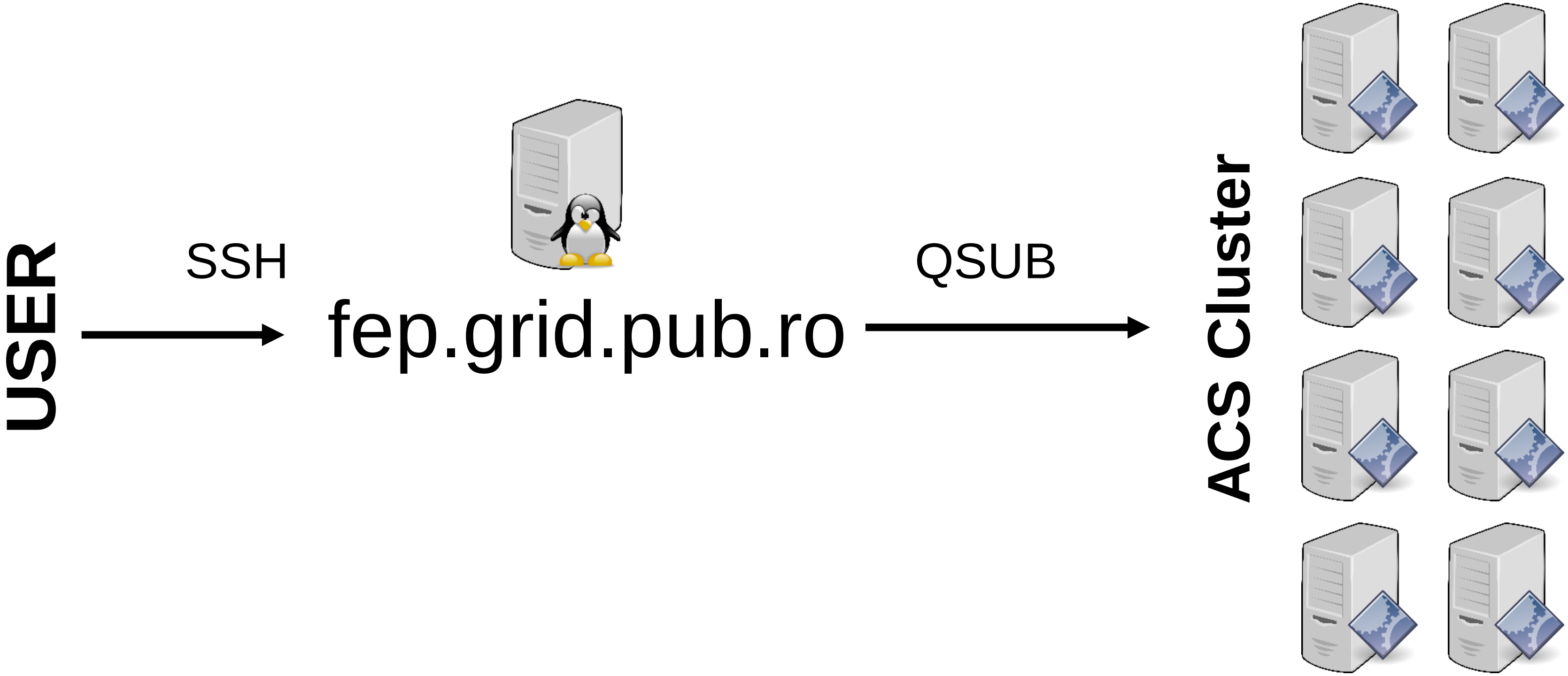
```
#include<mpi.h>
#include<stdio.h>

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Hello from 1/4

MPI memory

- There is no such thing as shared memory in MPI
- All variables are local per process (equivalent to private in OpenMP)
- To get information from one process to the other you have to use explicit communication
  - Send/Recv
  - Broadcast
  - Scatter
  - Gather

Running on the cluster



USER

SSH

fep.grid.pub.ro

QSUB

ACS Cluster

Account

cs.curs.pub.ro
Account

name.familyName

QSUB

Parameters

-cwd (change working directory)
-q queue_name (select a queue)

Run on fep.grid.pub.ro:

```
qsub –cwd –pe openmpi*1 <nTasks> <<EOF
> module load libraries/openmpi-1.6-gcc-4.7.0
> mpirun –n <nTasks> ./executable_name
> EOF
```

Current Working Directory
Makes sure the input/output files are taken/placed in the current directory.

Run on fep.grid.pub.ro:

```
qsub –cwd –pe openmpi*1 <nTasks> <<EOF
> module load libraries/openmpi-1.6-gcc-4.7.0
> mpirun –n <nTasks> ./executable_name
> EOF
```

Specify how many machines/tasks to use

Run on fep.grid.pub.ro:

```
qsub –cwd –pe openmpi*1 <nTasks> <<EOF
> module load libraries/openmpi-1.6-gcc-4.7.0
> mpirun –n <nTasks> ./executable_name
> EOF
```

Bash shorthand tells qsub a script is following

The script is surrounded by EOF

Using a bash script bash script:

```
qsub –cwd –pe openmpi*1 <nTasks> script.sh
```

script.sh contents:

```
#!/bin/bash
module load libraries/openmpi-1.6-gcc-4.7.0
mpirun –n <nTasks> ./executable_name
```