

Working Manual: High Performance Computing Facility (Fish@CMFRI)



ICAR-Central Marine Fisheries Research Institute
(Department of Agricultural Research and Education, Government of India)
P.B. No. 1603, Ernakulam North P.O., Kochi - 682 018, Kerala, India





Working Manual: High Performance Computing Facility (Fish@CMFRI)
CMFRI Training Manual Series No.33/2024

Published by:

Dr. A. Gopalakrishnan

Director

ICAR-Central Marine Fisheries Research Institute (ICAR-CMFRI)

P.B.NO.1603, Ernakulum North P.O., Kochi- 682018, Kerala, India

www.cmfri.org.in, Email: director.cmfri@icar.gov.in

Tel. No. +91-484-2394867; Fax. No. +91-484-2394909

Compiled & Edited by:

Eldho Varghese, J Jayasankar, Manu V K, Grinson George and Thejus T V

Cover Design: Abhilash P R

Publication, Production & Co-ordination:

Arun Surendran, Library & Documentation Centre, CMFRI

©2024 ICAR-Central Marine Fisheries Research Institute, Kochi

All rights reserved. Material contained in this publication may not be reproduced in any form without permission of the publisher.

Suggested Citation: Eldho Varghese, J Jayasankar, Manu V K, Grinson George and Thejus T V (2024) Working Manual: High Performance Computing Facility (Fish@CMFRI), CMFRI Training Manual Series No.33/2024, ICAR-Central Marine Fisheries Research Institute, Kochi, 50p.

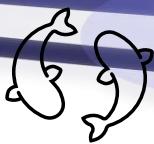


Working Manual: High Performance Computing Facility **(Fish@CMFRI)**



ICAR-Central Marine Fisheries Research Institute
(Department of Agricultural Research and Education, Government of India)
P.B. No. 1603, Ernakulam North P.O., Kochi - 682 018, Kerala, India





Preface

The advent of high-performance computing has ushered in a new era of scientific exploration, enabling researchers to address complex and data-intensive challenges with remarkable speed, accuracy, and efficiency. This manual is meticulously crafted to serve as a comprehensive guide for scientists, research scholars, and students who will harness the cutting-edge capabilities of the High-Performance Computational Facility (FISH@CMFRI). Strategically installed at the Agricultural Knowledge Management Unit (AKMU) of ICAR-CMFRI, Kochi, this state-of-the-art facility epitomizes the integration of advanced computational infrastructure into the scientific research ecosystem.

Developed with partial financial support under the prestigious DST-REVIVAL project *REhabilitation of Vibrio Infested waters of VembanAd Lake: Pollution and Solution*, the FISH@CMFRI facility represents a transformative leap forward in computational power and versatility. It not only underscores a commitment to addressing pressing environmental challenges but also exemplifies a broader vision to elevate the capabilities of the academic and research community, fostering excellence in computational science and innovation.

This facility is poised to catalyse research endeavours, significantly benefiting the academic and scientific community. Furthermore, it plays a pivotal role in capacity building by fostering a workforce skilled in high-performance computing techniques. By integrating cutting-edge emerging technologies, this facility also promotes innovation in research and teaching, laying the foundation for transformative scientific advancements.

We hope this manual serves as a valuable resource for all users, enhancing their experience with the facility and supporting their pursuit of excellence in research.

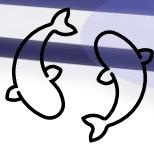
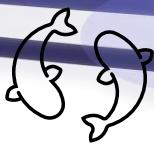


Table of Contents

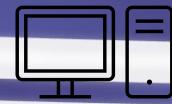
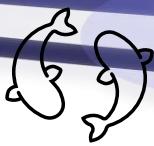
Sl. No.	Description	Page No.
1	Introduction	6
2	Hardware Specification	7
3	FISH@CMFRI	8
4	Basic Linux Commands	10
5	Job Scheduler	11
6	How to Access HPC	12
7	How to Run C Code in HPC	14
8	How to Run R Code in HPC	20
9	Transfer Files between Windows and HPC	27
10	Good Practices for Using HPC	35
11	FISH@CMFRI Power Off Procedure	36
12	FISH@CMFRI Power On Procedure	37
13	Installing Packages in Compute Node Image	39
14	Appendix	40
15	Additional Resources	46



Introduction

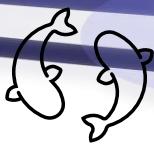
High-speed computing has been the "go to" option for many institutions doing research and development in biological science, especially natural resources. With the platform that usually hosts such research initiatives being multipronged and multi-sourced, data collation and analysis gets even more computer intensive. In any science-based initiative, it is next to impossible to have a data template which is homogenous, equispaced and equidistributed. More so for any other initiative that could be planned in marine fisheries scenarios. The inputs could be of any kind from organized tabulated data to sudden pieces of messages leading to actionable information akin to the Internet of Things and more rigorous systematic input from concerted exercises lie downscaling from macro-regional models like the ROMs.

All these needs both vertically sequenced and laterally accommodative computational architecture that could receive, preprocess, and store with equal efficacy. So in such situations, that are a scale above workstations bordering high-performance computing clusters would be the single-stop game changer solution. A high-performance computing facility (FISH@CMFRI) with multi-core computing and expansive scalability of big data analytics/streaming data processing architecture was developed at the Agriculture Knowledge Management Unit (AKMU) of ICAR-CMFRI Kochi.



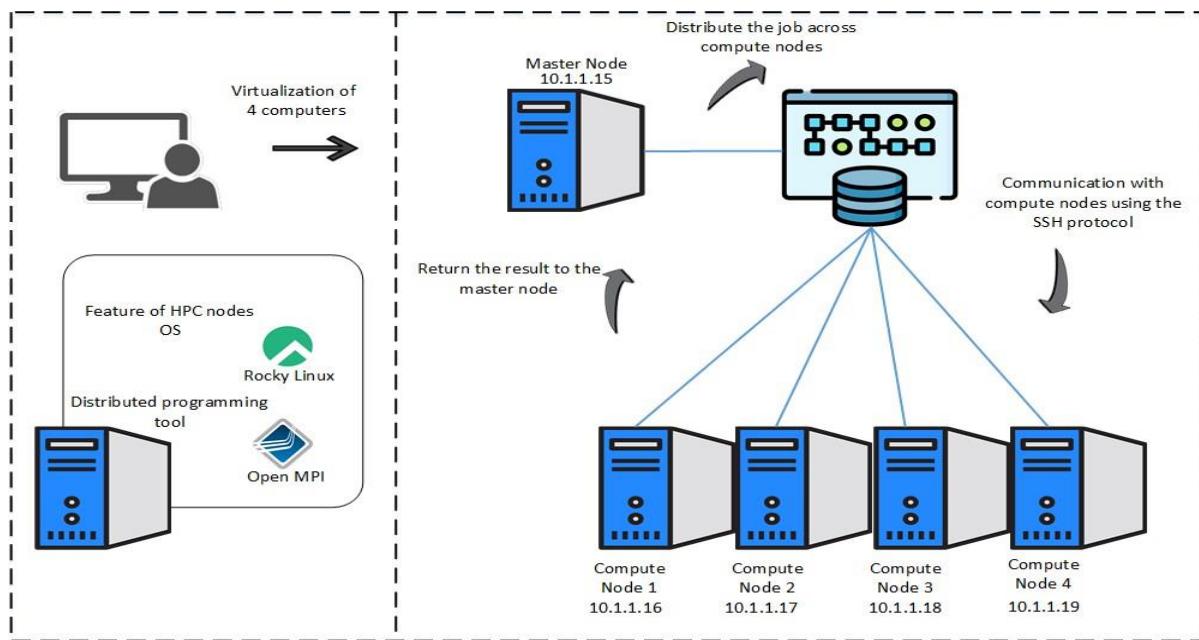
Hardware Specification

Computing Capacity	3 TFLOPS
No. of Compute Nodes	4
Total No. of Processors	8
Total No. of Cores	96
Memory per node	64 GB
Total RAM	256 GB
Total Usable Storage Capacity	4 TB
Primary Interconnect	10 Gbps Ethernet
Administration & Management Network	1 Gbps Ethernet
Operating System	Rocky Linux 8.8

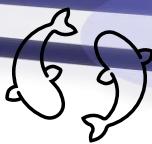


FISH@CMFRI

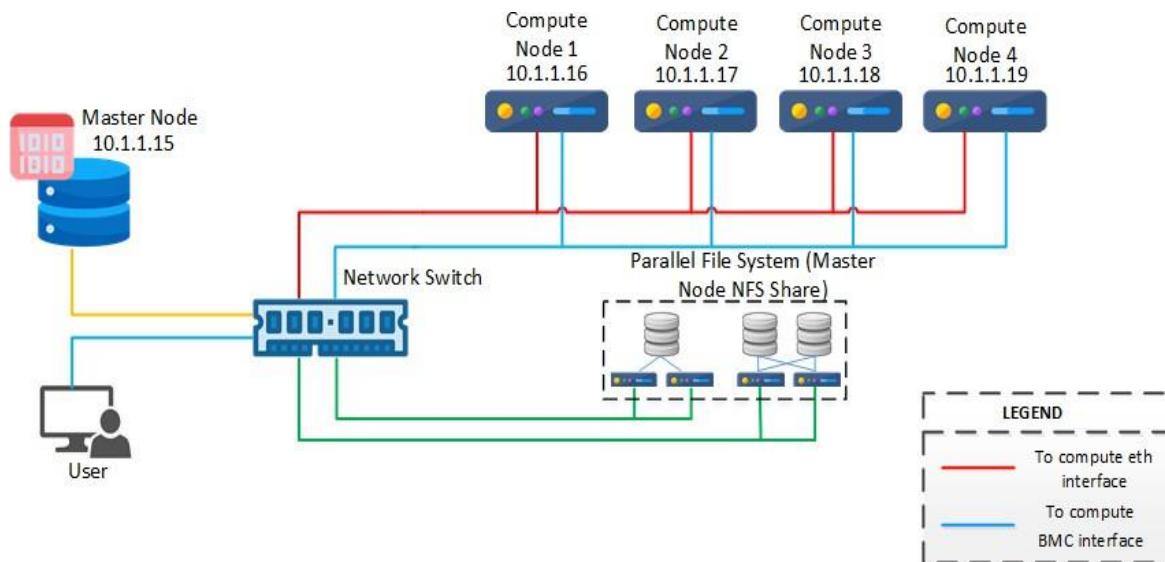
FISH@CMFRI HPC solution implemented to perform research and development activities with Rocky Linux 8.8 and OpenHPC stack.



OpenHPC is a collaborative, community effort initiated from a desire to aggregate a number of common ingredients required to deploy and manage High-Performance Computing (HPC) Linux clusters including provisioning tools, resource management, I/O clients, development tools, and a variety of scientific libraries. Packages provided by OpenHPC have been pre-built with HPC integration in mind with a goal to provide reusable building blocks for the HPC community. The master host connected to the local data center network is used to provision and manage the cluster backend. All 4 compute nodes are connected to the master node with local data center switch and these nodes are PXE booted with Rocky Linux 8.8 OS from the master node. Also, there is a shared NFS drive from the master node for running parallel processes across all nodes.



Configuration of FISH@CMFRI



1. User Details

User Name: user01

User Name: user02

User Name: user03

User Name: user04

User Name: user05

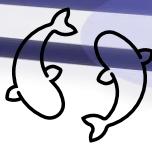
(Passwords will be provided on demand).

2. Nodes

Management Node	10.1.1.15
Compute-node1	10.1.1.16
Compute-node2	10.1.1.17
Compute-node3	10.1.1.18
Compute-node4	10.1.1.19

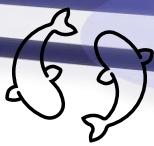
3. Compiler Details

Compiler: OpenMPI 4.1.5



Basic Linux commands

Command	Description
ls	The ls command displays list of files in human readable format.
rm	The rm command Remove files or directories.
cp	The cp command copies file from source to destination preserving same mode.
cd	The cd command (change directory) takes us to the destination directory.
cat	cat command can be used to display the contents of a single or multiple files
pwd	pwd command return the present working directory
mkdir	Create a folder or a directory.
rmdir	Delete a folder or a directory.
touch	The touch command is used to create a file. It can be anything, from an Empty txt file to an empty zip file. For example, "touch new.txt".
cp	Use the cp command to copy files through the command line.
mv	Use the mv command to move files through the command line.
locate	The locate command is used to locate a file in a Linux system.
nano, vi	nano and vi are already installed text editors in the Linux command line.
sudo	A widely used command in the Linux command line, sudo stands for "Super User Do". So, if you want any command to be done with administrative or root privileges, you can use the sudo command.
df	Use the df command to see the available disk space in each of the partitions in your system.
ping	Use ping to check your connection to a server.
Ctrl+C Ctrl+Z	Ctrl+C can be used to stop any command in terminal safely. If it doesn't stop with that, then Ctrl+Z can be used to force stop it.
clear	You can use the clear command to clear the terminal if it gets filled up with too many commands.
zip, unzip	Use zip to compress files into a zip archive, and unzip to extract files from a zip archive.



Job Scheduler

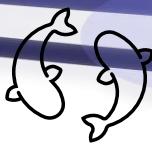
The HPC Job Scheduler Service is a software program that runs on the head/management node of a high-performance computing (HPC) cluster. It manages the jobs and tasks that are submitted to the cluster, allocating resources to them and monitoring their progress. You can configure the HPC Job Scheduler Service to control how resources are allocated and how jobs are handled. The job scheduler used at CMFRI is PBS pro. PBS stands for Portable Batch System, and it is a software system that is used to manage jobs on HPC clusters.

To run a parallel job using PBS script, you first need to create a PBS script. A PBS script is a text file that contains instructions for the PBS job scheduler. The PBS script will specify the following:

- The name of the job
- The number of nodes that the job needs
- The number of processors per node
- The amount of memory that the job needs
- The command that the job should run

Once you have created the PBS script, you can submit it to the PBS job scheduler using the `qsub` command. The `qsub` command will queue the job and the PBS job scheduler will start running it as soon as a set of nodes with the required resources becomes available.

The PBS job scheduler will monitor the progress of the job and will send you email notifications if the job fails or if it exceeds its allotted resources. When the job is finished, the PBS job scheduler will send you a notification that the job has completed.



How to Access HPC

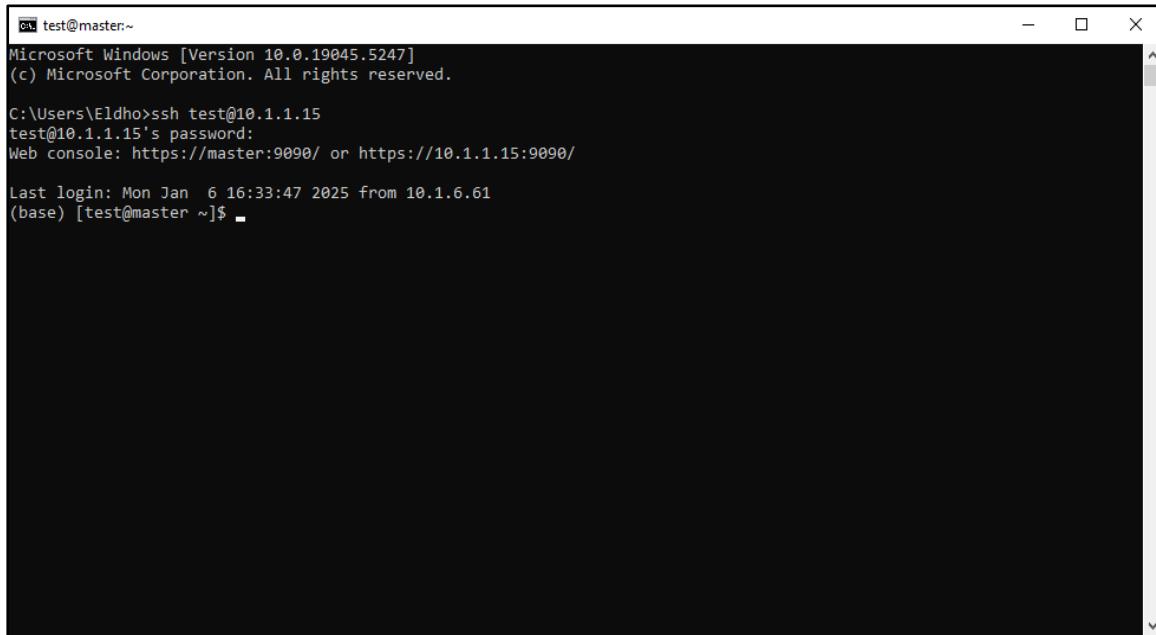
To Access or login to the HPC can be done using SSH (Secure Shell). You can use software like Putty or directly access SSH from Terminal / command prompt.

1. Connecting from Terminal/ Command line

Open Command Prompt, type

```
ssh username@10.1.1.15
```

When prompted enter your password and press Enter again. If the connection is successful, you will see a window like this.



```
cmd test@master:~  
Microsoft Windows [Version 10.0.19045.5247]  
(c) Microsoft Corporation. All rights reserved.  
C:\Users\Eldho>ssh test@10.1.1.15  
test@10.1.1.15's password:  
Web console: https://master:9090/ or https://10.1.1.15:9090/  
Last login: Mon Jan  6 16:33:47 2025 from 10.1.6.61  
(base) [test@master ~]$ -
```

* Here **test** is the username, you must enter your username.

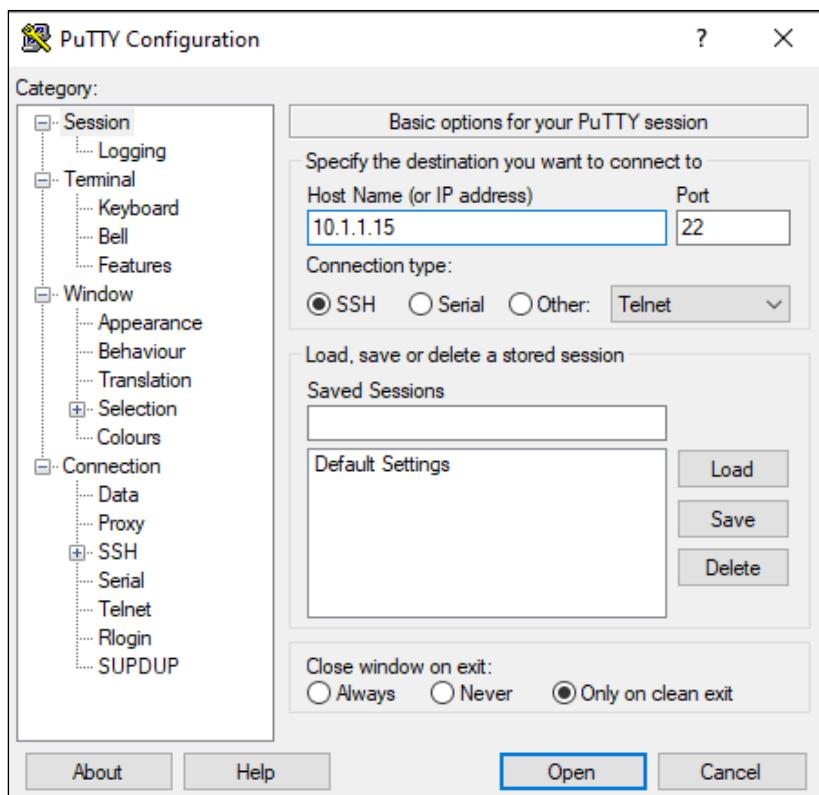
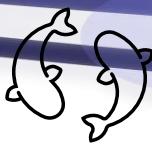
2. Connect using Putty

* If you already have access via Command Prompt, you can skip this step.

Putty software is available in Microsoft store; the link for the software is given below.

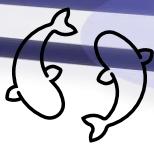
<https://apps.microsoft.com/detail/xpfnzksklbp7rj?hl=en-US&gl=IN>

After Installation open the App and type the host name as 10.1.1.15 and click open



If any security alert comes you can either choose to *Accept* or *Connect once*. After that enter your username and password.

```
test@master:~  
login as: test  
test@10.1.1.15's password:  
Web console: https://master:9090/ or https://10.1.1.15:9090/  
  
Last failed login: Mon Jan  6 17:15:56 IST 2025 from 10.1.1.1 on ssh:notty  
There were 13 failed login attempts since the last successful login.  
Last login: Mon Jan  6 16:39:16 2025 from 10.1.6.61  
(base) [test@master ~]$
```



How to Run C Code in HPC

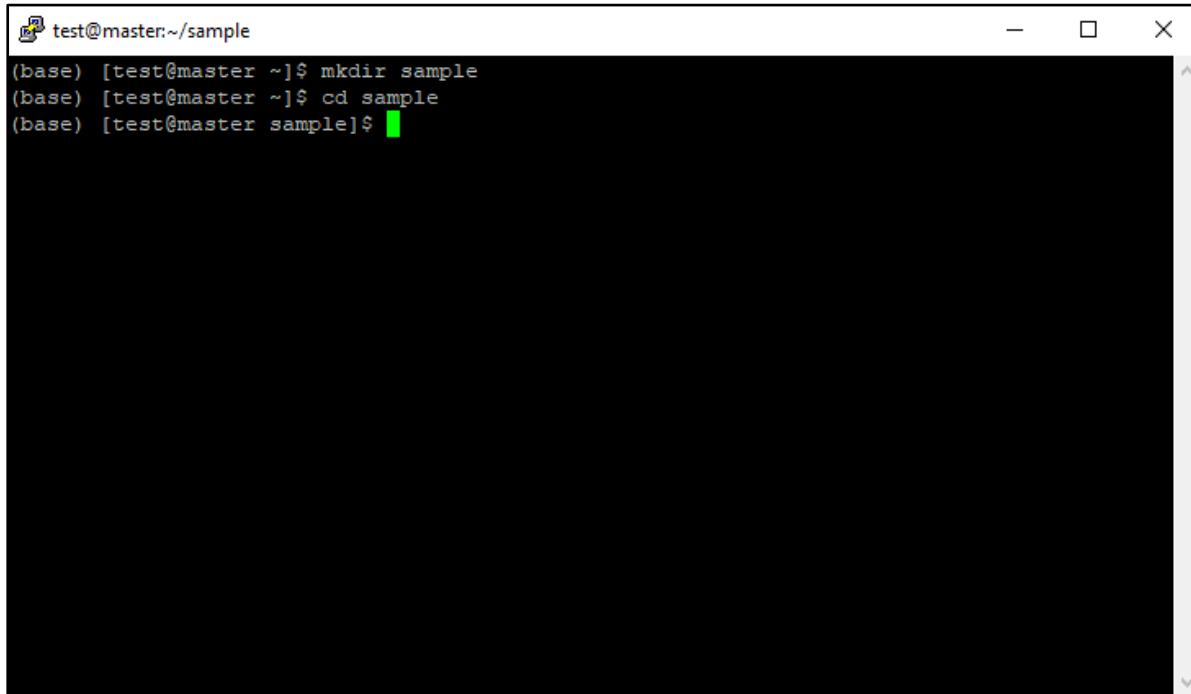
Creating a C program file

C program files can be created using editors such as vi or nano. Before creating a file first create a folder using the command.

```
mkdir foldername
```

To enter the folder type

```
cd foldername
```

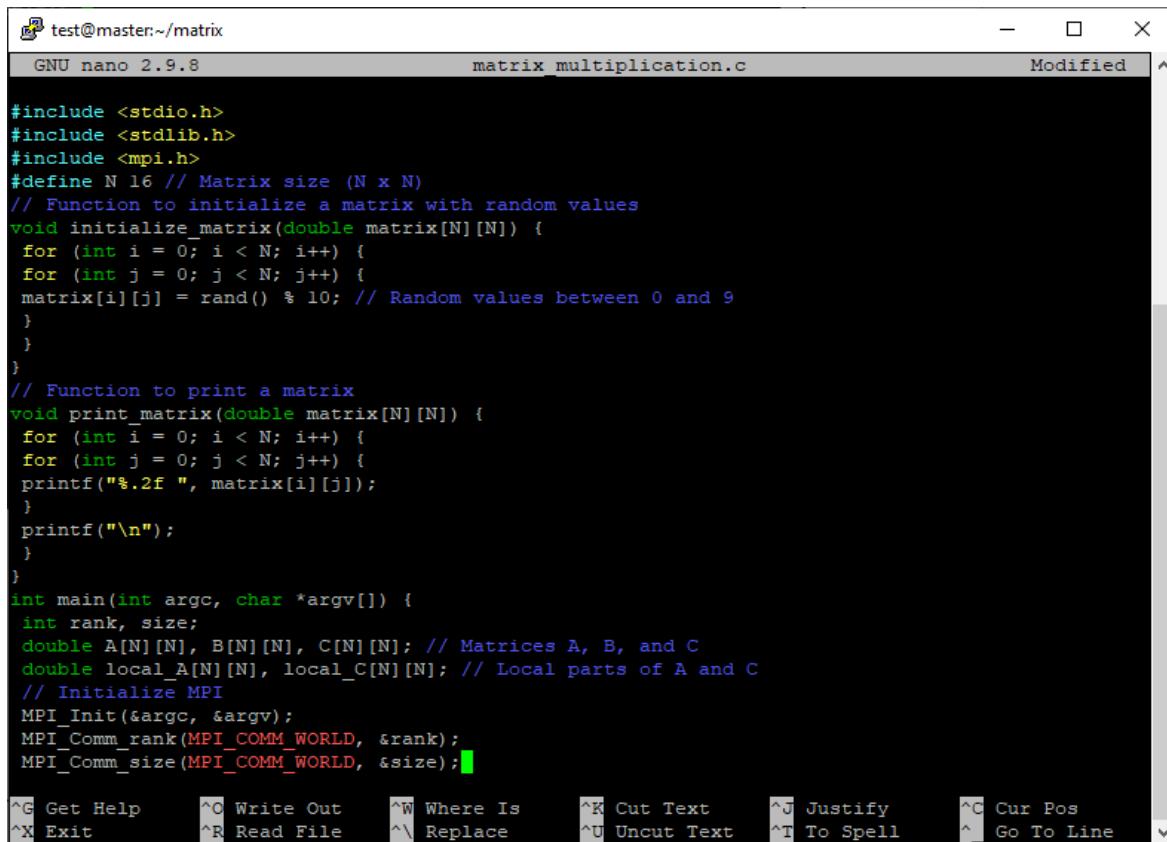
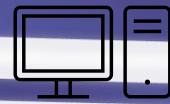
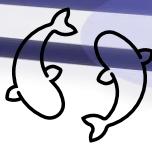


```
test@master:~/sample
(base) [test@master ~]$ mkdir sample
(base) [test@master ~]$ cd sample
(base) [test@master sample]$
```

To create a c program file type

```
nano matrix_multiplication.c
```

A text editor will open like this, here you can either type your code or copy code from somewhere else. Just press right click to paste your code.

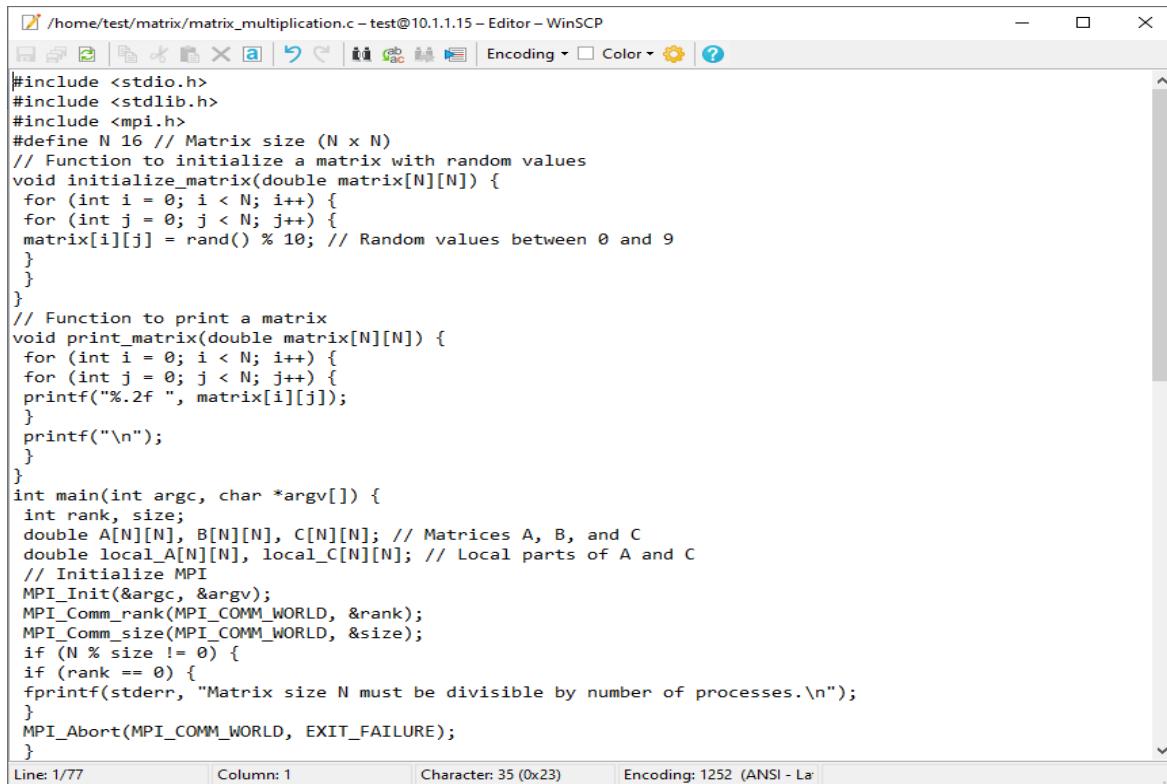


```
test@master:~/matrix
GNU nano 2.9.8 matrix_multiplication.c Modified

#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define N 16 // Matrix size (N x N)
// Function to initialize a matrix with random values
void initialize_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = rand() % 10; // Random values between 0 and 9
        }
    }
}
// Function to print a matrix
void print_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%.2f ", matrix[i][j]);
        }
        printf("\n");
    }
}
int main(int argc, char *argv[]) {
    int rank, size;
    double A[N][N], B[N][N], C[N][N]; // Matrices A, B, and C
    double local_A[N][N], local_C[N][N]; // Local parts of A and C
    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

^G Get Help      ^O Write Out      ^W Where Is      ^K Cut Text      ^J Justify      ^C Cur Pos
^X Exit          ^R Read File      ^\ Replace       ^U Uncut Text    ^T To Spell      ^L Go To Line
```

To save the code press **Ctrl+S** and to Exit the editor press **Ctrl+X**.

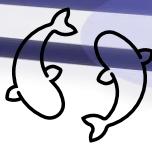


```
/home/test/matrix/matrix_multiplication.c - test@10.1.1.15 - Editor - WinSCP

#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define N 16 // Matrix size (N x N)
// Function to initialize a matrix with random values
void initialize_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = rand() % 10; // Random values between 0 and 9
        }
    }
}
// Function to print a matrix
void print_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%.2f ", matrix[i][j]);
        }
        printf("\n");
    }
}
int main(int argc, char *argv[]) {
    int rank, size;
    double A[N][N], B[N][N], C[N][N]; // Matrices A, B, and C
    double local_A[N][N], local_C[N][N]; // Local parts of A and C
    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (N % size != 0) {
        if (rank == 0) {
            fprintf(stderr, "Matrix size N must be divisible by number of processes.\n");
        }
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
    }
}

Line: 1/77 Column: 1 Character: 35 (0x23) Encoding: 1252 (ANSI - La
```

Full code available in the appendix section.



A C program must be compiled before running. To compile the code, type

```
mpicc -o matrix_multiplication matrix_multiplication.c
```

mpicc - This is the MPI compiler wrapper for C programs. It ensures that your program is compiled with the appropriate MPI libraries and headers.

-o matrix_multiplication - The -o flag specifies the output filename for the compiled program. In this case, the compiled binary will be named matrix_multiplication

matrix_multiplication.c: This is the input source file, i.e., the C code you wrote that contains the MPI program.

While it is possible to submit programs directly to HPC it is generally preferable to create a job script (PBS script). The job script is just like any other code that contains commands that the user would like to run. You can create a PBS script just like how we created c program file, please ensure that the script file has a “.pbs” extension.

Creating a PBS script file

```
nano matrix_multiplication.pbs
```

Sample PBS script is given below.

```
#!/bin/bash

#PBS -N matrix_mult

#PBS -l nodes=4:ppn=4

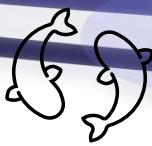
#PBS -l walltime=00:05:00

#PBS -j oe

#PBS -o matrix_mult_output.txt

cd $PBS_O_WORKDIR

mpirun -np 16 ./matrix_multiplication
```



#PBS -N matrix_mult - This sets the job name to matrix_mult. This name will appear in the job's output files and can be used to identify the job.

#PBS -l nodes=4:ppn=4 - This request 4 nodes, with 4 processors per node. This means the job will run on 4 compute nodes, each utilizing 4 CPU cores.

#PBS -l walltime=00:05:00 - This sets a maximum wall time of 5 minutes for the job. If the job exceeds this time, it will be terminated.

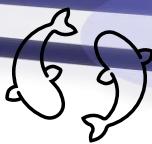
#PBS -j oe - This directs the standard output and error to the same file, meaning both stdout and stderr will be combined into the same file.

#PBS -o matrix_mult_output.txt - This specifies the output file name where the job's combined output (from stdout and stderr) will be stored, which in this case is matrix_mult_output.txt.

cd \$PBS_O_WORKDIR - This changes the working directory to the directory from which the job was submitted. \$PBS_O_WORKDIR is an environment variable that stores the path of the directory the script was run from.

mpirun -np 16 ./matrix_multiplication - This command runs the MPI program mpi_matrix_mult using 16 processes in total.

- **mpirun:** This is the command used to launch MPI jobs.
- **-np 16:** This option specifies the number of processes to run, in this case, 16 processes (4 node * 4 processors per node), if we use only 2 nodes and 5 processors per node then it will be 10.
- **./matrix_multiplication:** This is likely an executable that performs matrix multiplication using MPI for parallelization.



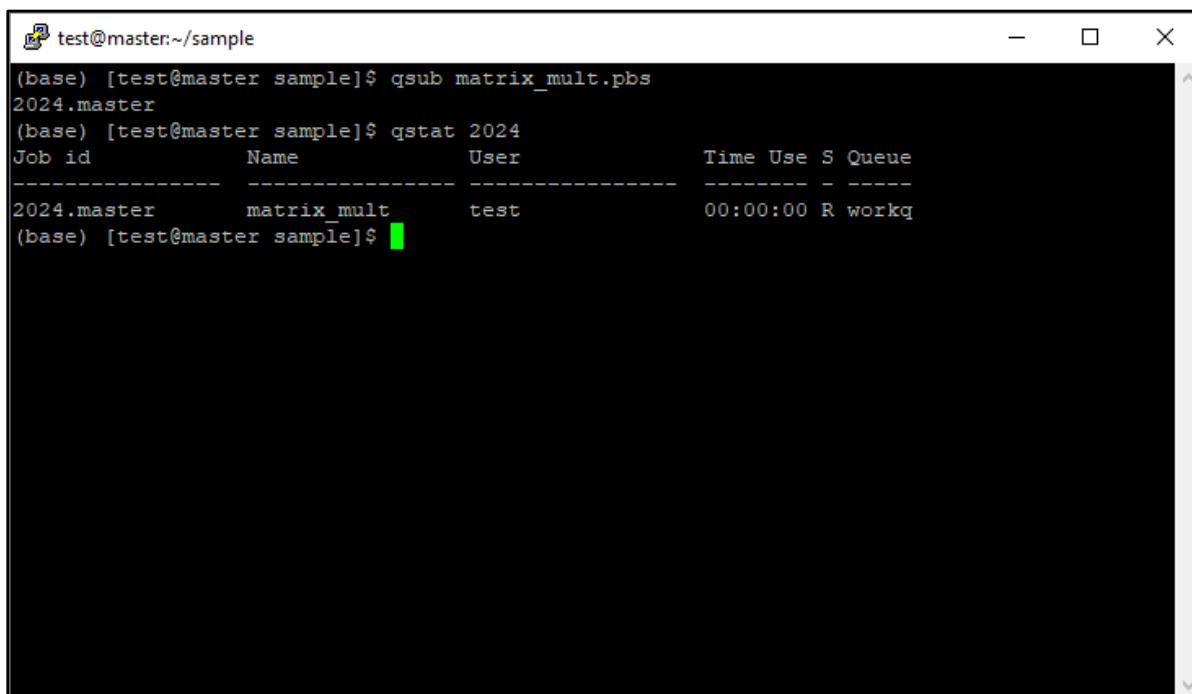
Users can submit their jobs by using qsub command followed by pbs script name.

```
qsub matrix_mult.pbs
```

User can also check the status of the submitted job using qstat command,

```
qstat <Job_id> or qstat -an
```

-an : Where **a** refers to status of all jobs in the pbs queue and **n** refers to which nodes the jobs are running



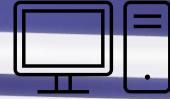
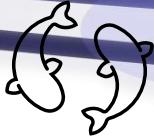
test@master:~/sample

```
(base) [test@master sample]$ qsub matrix_mult.pbs
2024.master
(base) [test@master sample]$ qstat 2024
Job id          Name      User          Time Use S Queue
-----  -----
2024.master    matrix_mult  test  00:00:00 R workq
(base) [test@master sample]$
```

*2024 is the Job id PBS assigned to the above job as you can see in the picture.

After the job completes, users can refer to the output files for result details or error diagnostics. For that type

```
cat matrix_mult_output.txt
```



```
test@master:~/sample$ cat matrix_mult_output.txt
(base) [test@master sample]$ cat matrix_mult_output.txt
/home/test/.bashrc: line 46: /usr/local/gromacs/bin/GMXRC: No such file or directory
Matrix A:
3.00 6.00 7.00 5.00 3.00 5.00 6.00 2.00 9.00 1.00 2.00 7.00 0.00 9.00 3.00 6.00
0.00 6.00 2.00 6.00 1.00 8.00 7.00 9.00 2.00 0.00 2.00 3.00 7.00 5.00 9.00 2.00
2.00 8.00 9.00 7.00 3.00 6.00 1.00 2.00 9.00 3.00 1.00 9.00 4.00 7.00 8.00 4.00
5.00 0.00 3.00 6.00 1.00 0.00 6.00 3.00 2.00 0.00 6.00 1.00 5.00 5.00 4.00 3.00
6.00 5.00 6.00 9.00 3.00 7.00 4.00 5.00 2.00 5.00 4.00 7.00 4.00 4.00 4.00 3.00
7.00 8.00 6.00 8.00 4.00 3.00 1.00 4.00 9.00 2.00 0.00 6.00 8.00 9.00 2.00
6.00 6.00 8.00 9.00 5.00 0.00 4.00 7.00 2.00 1.00 7.00 4.00 2.00 2.00 6.00
1.00 6.00 5.00 6.00 2.00 1.00 0.00 5.00 0.00 5.00 1.00 0.00 7.00 5.00 5.00
5.00 0.00 6.00 7.00 6.00 3.00 6.00 5.00 3.00 8.00 4.00 8.00 1.00 2.00 6.00
3.00 9.00 0.00 8.00 8.00 5.00 0.00 9.00 6.00 3.00 8.00 5.00 6.00 1.00 1.00 5.00
9.00 8.00 4.00 8.00 1.00 0.00 3.00 0.00 4.00 4.00 4.00 7.00 6.00 3.00 1.00
7.00 5.00 9.00 6.00 2.00 1.00 7.00 8.00 5.00 7.00 4.00 1.00 8.00 5.00 9.00 7.00
5.00 3.00 8.00 8.00 3.00 1.00 8.00 9.00 6.00 4.00 3.00 3.00 3.00 8.00 6.00 0.00
4.00 8.00 8.00 8.00 9.00 7.00 7.00 6.00 4.00 3.00 0.00 3.00 0.00 9.00 2.00 5.00
4.00 0.00 5.00 9.00 4.00 6.00 9.00 2.00 2.00 4.00 7.00 7.00 5.00 4.00 8.00 1.00
2.00 8.00 9.00 3.00 6.00 8.00 0.00 2.00 1.00 0.00 5.00 1.00 1.00 0.00 8.00 5.00

Matrix B:
0.00 6.00 4.00 6.00 2.00 5.00 8.00 6.00 2.00 8.00 4.00 7.00 2.00 4.00 0.00 6.00
2.00 9.00 9.00 8.00 1.00 0.00 3.00 0.00 1.00 0.00 4.00 0.00 3.00 9.00 1.00
9.00 6.00 0.00 6.00 2.00 0.00 5.00 6.00 0.00 4.00 0.00 0.00 3.00 0.00 0.00 0.00
6.00 0.00 5.00 6.00 3.00 6.00 7.00 2.00 0.00 2.00 5.00 0.00 6.00 0.00 2.00 7.00
6.00 1.00 3.00 2.00 1.00 5.00 9.00 9.00 1.00 4.00 5.00 1.00 0.00 0.00 7.00 5.00
7.00 0.00 4.00 8.00 0.00 4.00 2.00 9.00 6.00 1.00 0.00 4.00 2.00 2.00 2.00 0.00
5.00 5.00 2.00 9.00 0.00 2.00 9.00 3.00 8.00 0.00 4.00 0.00 9.00 1.00 9.00 6.00
2.00 5.00 4.00 9.00 9.00 3.00 6.00 0.00 5.00 0.00 2.00 9.00 4.00 3.00 5.00
1.00 7.00 4.00 3.00 1.00 4.00 6.00 9.00 4.00 2.00 2.00 6.00 4.00 1.00 2.00 8.00
8.00 9.00 2.00 0.00 8.00 8.00 6.00 8.00 3.00 8.00 3.00 3.00 3.00 8.00 0.00 4.00
7.00 6.00 8.00 9.00 0.00 6.00 8.00 7.00 9.00 6.00 0.00 4.00 1.00 0.00 4.00 8.00
6.00 5.00 2.00 6.00 5.00 8.00 7.00 9.00 6.00 0.00 4.00 1.00 0.00 4.00 8.00 7.00
0.00 8.00 6.00 2.00 4.00 7.00 9.00 3.00 9.00 2.00 8.00 3.00 0.00 1.00 7.00 8.00
9.00 1.00 5.00 4.00 9.00 2.00 5.00 7.00 4.00 9.00 9.00 4.00 5.00 9.00 3.00 5.00
7.00 0.00 8.00 1.00 9.00 9.00 7.00 8.00 2.00 5.00 3.00 4.00 9.00 0.00 2.00 0.00
1.00 9.00 6.00 2.00 1.00 2.00 0.00 7.00 3.00 1.00 9.00 0.00 5.00 6.00 7.00

Result Matrix C:
373.00 374.00 376.00 323.00 307.00 351.00 359.00 466.00 334.00 231.00 305.00 289.00 235.00 273.00 340.00 359.00
325.00 309.00 369.00 296.00 363.00 390.00 368.00 385.00 322.00 191.00 269.00 245.00 316.00 182.00 317.00 283.00
425.00 421.00 454.00 318.00 412.00 465.00 521.00 369.00 269.00 351.00 323.00 240.00 273.00 364.00 371.00
235.00 299.00 291.00 257.00 216.00 298.00 313.00 302.00 287.00 175.00 249.00 241.00 214.00 190.00 228.00 286.00
384.00 382.00 365.00 377.00 355.00 442.00 414.00 434.00 361.00 249.00 323.00 270.00 243.00 260.00 297.00 334.00
438.00 429.00 445.00 346.00 440.00 469.00 494.00 501.00 349.00 360.00 422.00 340.00 276.00 325.00 312.00 379.00
282.00 464.00 408.00 320.00 325.00 422.00 435.00 416.00 350.00 220.00 332.00 327.00 254.00 256.00 327.00 413.00
333.00 343.00 283.00 336.00 273.00 415.00 323.00 443.00 299.00 225.00 229.00 186.00 187.00 269.00 287.00 317.00
365.00 552.00 490.00 310.00 319.00 498.00 512.00 533.00 433.00 262.00 378.00 367.00 242.00 337.00 415.00 472.00
304.00 444.00 496.00 329.00 341.00 428.00 435.00 461.00 327.00 189.00 313.00 320.00 216.00 287.00 327.00 391.00
285.00 491.00 389.00 359.00 393.00 401.00 490.00 387.00 325.00 240.00 337.00 286.00 238.00 278.00 322.00 322.00
407.00 415.00 384.00 359.00 359.00 523.00 494.00 413.00 404.00 345.00 365.00 341.00 325.00 311.00 325.00 426.00
400.00 362.00 392.00 363.00 405.00 461.00 395.00 450.00 350.00 314.00 351.00 369.00 352.00 274.00 308.00 377.00
432.00 398.00 412.00 359.00 378.00 407.00 403.00 490.00 333.00 297.00 374.00 306.00 275.00 326.00 368.00 363.00
432.00 356.00 374.00 417.00 324.00 467.00 483.00 472.00 427.00 235.00 355.00 265.00 306.00 250.00 317.00 353.00
314.00 270.00 380.00 215.00 233.00 325.00 255.00 364.00 239.00 167.00 207.00 217.00 147.00 200.00 262.00 185.00

Time taken: 0.012920 seconds
(base) [test@master sample]$
```

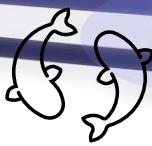
*Time Taken: 0.012920 seconds

*Please note that execution time may vary depending on system load, network conditions, and available resources.

The result above corresponds to a 16 x 16 matrix multiplication. Using the same setup, we achieved a time of 180.214546 seconds for a 10000 x 10000 matrix multiplication. The code used for this calculation is provided in the appendix section, where you can adjust the matrix size (N) up to 10000 to observe the results.

To cancel a Job

qdel <Job_id>



How to Run R Code in HPC

After successfully logging in, you can create an R script just like how we created a C program file. Ensure the script file has an “.R” extension. Unlike a C program, R code does not require compilation. Simply create the R script and a PBS script to run the R code on the HPC.

sample.pbs script is given below.

```
#!/bin/bash
#PBS -N R_MPI_parallel_test
#PBS -l nodes=4:ppn=24,mem=64gb
#PBS -l walltime=02:00:00
#PBS -j oe
#PBS -o output.log

module load R
module load gnu12/12.3.0
module load openmpi4/4.1.6

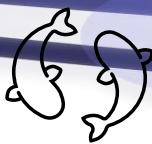
export R_LIBS="/home/test/R/x86_64-redhat-linux-gnu-library/4.4:$R_LIBS"

cd $PBS_O_WORKDIR

mpirun --map-by node -np 96 Rscript sample.R
```

**sample.R script is provided in the appendix section.*

module load <module-name> - This command loads the necessary module into your environment on the HPC. For parallel execution in R above three modules are required. It loads R environment, GNU Compiler Collection, which might be required for dependencies, OpenMPI for parallel execution.



export R_LIBS="/home/test/R/x86_64-redhat-linux-gnu-library/4.4:\$R_LIBS" - Adds a user-specific R package directory to the R library search path.

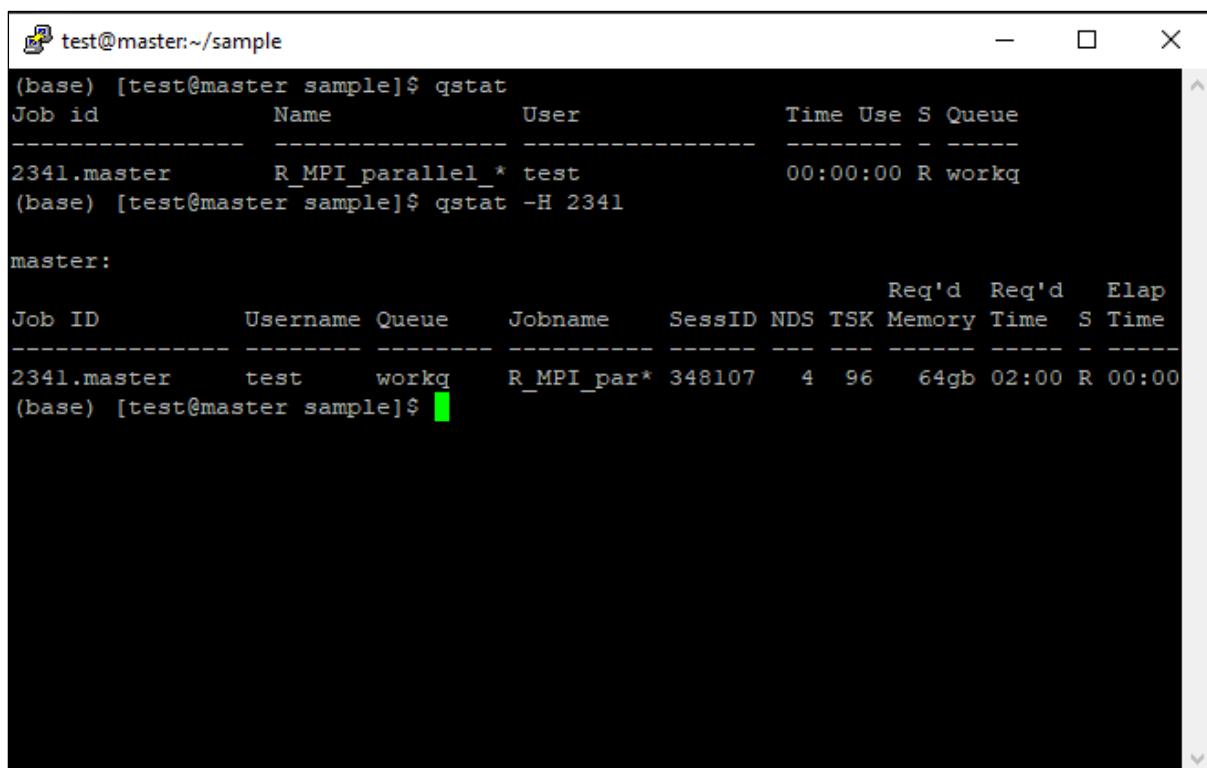
mpirun --map-by node -np 96 Rscript sample.R - Runs the R script named *sample.R*, “*--map-by node*” command will force MPI to distribute ranks evenly across nodes.

Users can submit their jobs by using qsub command followed by pbs script name.

```
qsub sample.pbs
```

To check the status of the specific job in details type,

```
qstat -H <Job_id>
```

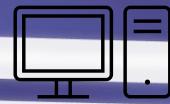
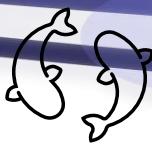


```
test@master:~/sample
(base) [test@master sample]$ qstat
Job id          Name          User          Time Use S Queue
-----
2341.master    R_MPI_parallel_* test          00:00:00 R workq
(base) [test@master sample]$ qstat -H 2341

master:
          Req'd  Req'd   Elap
Job ID      Username Queue   Jobname   SessID NDS TSK Memory Time   S Time
-----
2341.master  test      workq  R_MPI_par* 348107  4  96   64gb 02:00 R 00:00
(base) [test@master sample]$
```

The 'S' column represents the status code of the job, with 'R' indicating that the job is Running. A complete list of status codes can be found in the appendix. After the job finishes, multiple files will be generated based on your code, which can be viewed using the **ls** command. To open a specific file, you can use the **cat** command.

```
cat output.log
```



```
test@master:~/sample
(base) [test@master sample]$ qstat
Job id          Name          User          Time Use S Queue
-----  -----
2341.master     R_MPI_parallel_* test          06:45:44 R workq
(base) [test@master sample]$ qstat -H 2341

master:
          Req'd  Req'd   Elap
Job ID      Username Queue   Jobname  SessID NDS TSK Memory Time  S Time
-----  -----
2341.master   test     workq   R_MPI_par* 348107  4  96   64gb 02:00 R 00:05
(base) [test@master sample]$ qstat -H 2341

master:
          Req'd  Req'd   Elap
Job ID      Username Queue   Jobname  SessID NDS TSK Memory Time  S Time
-----  -----
2341.master   test     workq   R_MPI_par* 348107  4  96   64gb 02:00 F 00:11
(base) [test@master sample]$ ls
matrix_results.rds  output.log  sample.pbs  sample.R
(base) [test@master sample]$ cat output.log
[1] "Time taken: 11.0130951841672"
(base) [test@master sample]$
```

*Time Taken: 11.0130951841672

Above R code performs intensive matrix operations and statistical analysis that go beyond the capabilities of a standard computer. If you check the code which is in appendix section, the computed results are saved as an RDS file (matrix_results.rds) for further analysis, which will be discussed in later sections.

In **R** Parallel Packages like *doParallel*, *parallel*, *pbdMPI*, *doMPI* and *doFuture* uses a common set of instructions to use parallel capabilities as follows:

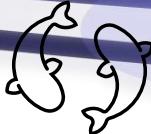
```
library("package-name")

cl <- makeCluster(NumberOfCores)

register_cluster(cl)

... #code to be run in parallel mode

stopCluster(cl)
```



To verify whether the code is running across all four nodes, the **htop** command can be used. It's a popular interactive process monitoring tool in Linux. It provides a real-time overview of system resource usage, including CPU, memory, and running tasks. The below screenshot shows our four compute nodes.

```
test@compute-node1:~
```

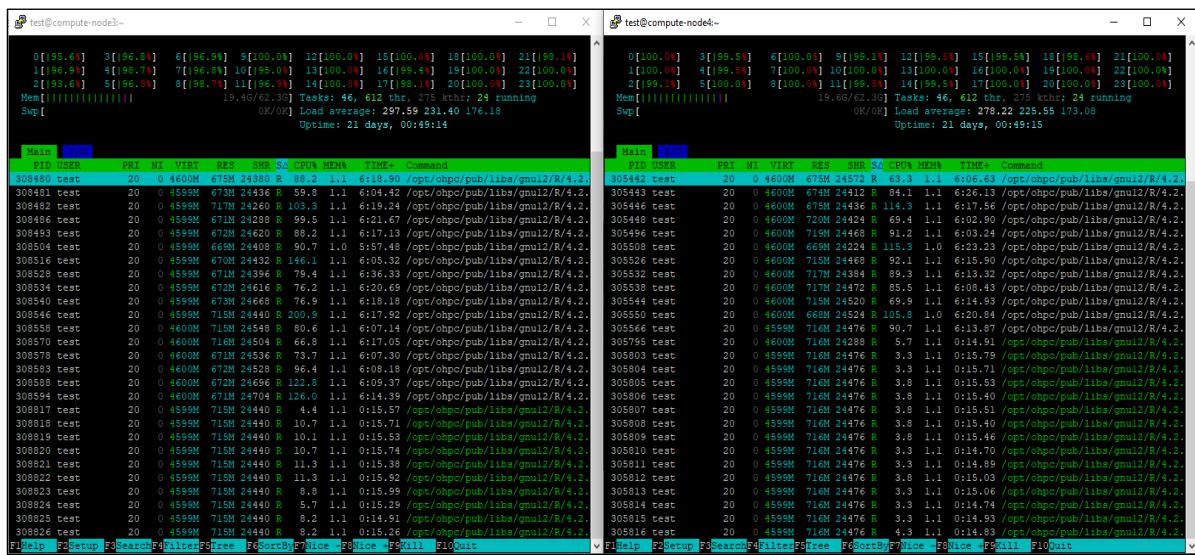
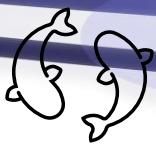
Main [1/2]									
PID	USER	PRI	NI	VIRT	RES	SHR	SC	CPU%	MEM%
395976	test	20	0	4599M	671M	24472	R	68.6	1.1
395978	test	20	0	4599M	718M	24556	R	116.9	1.1
395989	test	20	0	4599M	671M	24216	R	89.6	1.1
396002	test	20	0	4599M	716M	24540	R	88.9	1.1
396007	test	20	0	4599M	671M	24560	R	78.8	1.1
396014	test	20	0	4599M	715M	24716	R	61.6	1.1
396020	test	20	0	4599M	671M	24608	R	116.9	1.1
396039	test	20	0	4599M	717M	24608	R	149.3	1.1
396044	test	20	0	4599M	716M	24572	R	157.4	1.1
396050	test	20	0	4599M	716M	24656	R	46.4	1.1
396082	test	20	0	4600M	715M	24560	R	111.8	1.1
396086	test	20	0	4600M	715M	24428	R	78.1	1.1
396311	test	20	0	4599M	671M	24216	R	3.8	1.1
396312	test	20	0	4599M	671M	24216	R	3.8	1.1
396313	test	20	0	4599M	671M	24216	R	3.8	1.1
396314	test	20	0	4599M	671M	24216	R	4.4	1.1
396315	test	20	0	4599M	671M	24216	R	3.8	1.1
396316	test	20	0	4599M	671M	24216	R	3.8	1.1
396317	test	20	0	4599M	671M	24216	R	3.8	1.1
396318	test	20	0	4599M	671M	24216	R	3.8	1.1
396319	test	20	0	4599M	671M	24216	R	3.8	1.1
396320	test	20	0	4599M	671M	24216	R	3.8	1.1
396321	test	20	0	4599M	671M	24216	R	3.8	1.1
396322	test	20	0	4599M	671M	24216	R	4.4	1.1
396324	test	20	0	4599M	671M	24216	R	3.8	1.1
396325	test	20	0	4599M	671M	24216	R	3.8	1.1
396326	test	20	0	4599M	671M	24216	R	3.8	1.1
396327	test	20	0	4599M	671M	24216	R	3.8	1.1
396328	test	20	0	4599M	671M	24216	R	3.2	1.1
396329	test	20	0	4599M	671M	24216	R	3.8	1.1
396330	test	20	0	4599M	671M	24216	R	3.8	1.1
396331	test	20	0	4599M	671M	24216	R	3.8	1.1
396332	test	20	0	4599M	671M	24216	R	3.8	1.1
396335	test	20	0	4599M	671M	24472	R	3.2	1.1

```
Mem[1/1] 19.6G/62.3G Tasks: 53, 613 thr, 276 kthr; 24 running
Swap[0/0] OK/OK Load average: 220.97 114.08 126.34
Uptime: 21 days, 00:44:17
F1Help F2Setup F3Search F4Filter F5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit
```

```
test@compute-node2:~
```

Main [1/2]									
PID	USER	PRI	NI	VIRT	RES	SHR	SC	CPU%	MEM%
309586	test	20	0	4600M	715M	24800	R	83.5	1.1
309598	test	20	0	4600M	671M	24556	R	95.5	1.0
309615	test	20	0	4600M	715M	24480	R	72.7	1.1
309621	test	20	0	4600M	699M	24340	R	86.6	1.0
309627	test	20	0	4600M	715M	24624	R	210.0	1.1
309639	test	20	0	4600M	717M	24612	R	115.7	1.1
309651	test	20	0	4600M	717M	24692	R	81.6	1.1
309663	test	20	0	4600M	716M	24516	R	165.7	1.1
309681	test	20	0	4600M	716M	24540	R	84.1	1.1
309696	test	20	0	4600M	699M	24480	R	159.4	1.0
309726	test	20	0	4600M	716M	24480	R	3.8	1.1
309731	test	20	0	4600M	716M	24480	R	4.4	1.1
309735	test	20	0	4600M	716M	24480	R	4.4	1.1
309740	test	20	0	4600M	716M	24480	R	3.8	1.1
309745	test	20	0	4600M	716M	24480	R	3.8	1.1
309746	test	20	0	4600M	716M	24480	R	3.2	1.1
309748	test	20	0	4600M	717M	24692	R	3.8	1.1
309749	test	20	0	4600M	717M	24692	R	4.4	1.1
309750	test	20	0	4600M	717M	24692	R	3.2	1.1
309751	test	20	0	4600M	717M	24692	R	3.8	1.1
309752	test	20	0	4600M	717M	24692	R	3.8	1.1
309753	test	20	0	4600M	717M	24692	R	3.2	1.1
309754	test	20	0	4600M	717M	24692	R	3.8	1.1
309755	test	20	0	4600M	717M	24692	R	3.2	1.1
309756	test	20	0	4600M	717M	24692	R	3.2	1.1
309757	test	20	0	4600M	717M	24692	R	3.8	1.1
309758	test	20	0	4600M	717M	24692	R	3.2	1.1
309759	test	20	0	4600M	717M	24692	R	3.2	1.1
309760	test	20	0	4600M	717M	24692	R	3.2	1.1
309761	test	20	0	4600M	717M	24692	R	3.2	1.1
309762	test	20	0	4600M	717M	24692	R	3.2	1.1
309763	test	20	0	4600M	717M	24692	R	3.2	1.1
309764	test	20	0	4600M	717M	24692	R	3.8	1.1
309765	test	20	0	4600M	717M	24692	R	3.2	1.1

```
Mem[1/1] 19.8G/62.3G Tasks: 50, 612 thr, 277 kthr; 24 running
Swap[0/0] OK/OK Load average: 260.77 145.08 138.20
Uptime: 21 days, 00:45:23
F1Help F2Setup F3Search F4Filter F5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit
```



* Logging in to each node is required to view resource usage using the htop command

Explanation of the key elements in the image

CPU Usage: The numbers 0-23 indicate the logical CPU cores (24 cores total). Each bar represents the usage of an individual core, with 100.0% indicating full utilization.

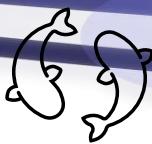
Memory Usage: The Mem bar shows the amount of system memory used. The numbers 24.0G/62.3G indicate that 24 GB out of 62.3 GB total memory is currently in use.

Swap Usage: The Swp section indicates swap memory usage. 0K/0K suggests that no swap space is allocated or in use.

Task Information: Tasks: 50 indicates there are 50 total processes running. 612 thr refers to the total number of threads across all processes. 24 running means 24 processes are actively using the CPU.

Load Average: The values 175.82 263.15 180.74 represent system load averages over the past 1, 5, and 15 minutes, respectively. These values are significantly high, indicating the system is under heavy CPU load.

System Uptime: Uptime: 21 days, 00:34:46 shows that the system has been running continuously for 21 days and 34 minutes.



Running R Code in Parallel on a Single Node

In R, packages like *parallel* and *doParallel* provide efficient parallel computing capabilities, but they are limited to a single node. These packages utilize multi-core processing by dividing tasks across the available cores of a single machine, enabling significant performance improvements for data-intensive computations. However, they do not support distributed computing across multiple nodes in an HPC cluster.

For workloads that require multi-node parallelism, R users need to leverage specialized frameworks such as **MPI (Message Passing Interface)** through packages like *pbdMPI* and *Rmpi* like we used before. R program that uses only single node is given below.

```
#!/home/test/singlecore/sample.R – test@10.1.1.15 – Editor – WinSCP
File Edit View Insert Cell Run Cell Help Encoding Color ? 
library(parallel)
library(doParallel)

size <- 16
cores <- 20

# Setup local cluster
cl <- makeCluster(cores)
registerDoParallel(cl)

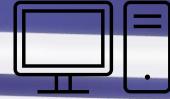
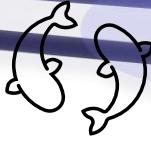
# Create matrices
A <- matrix(runif(size^2), size)
B <- matrix(runif(size^2), size)

start <- Sys.time()
chunks <- ceiling(size/cores)
result <- foreach(i = seq(1, size, by=chunks), .combine='rbind') %dopar% {
  end_idx <- min(i + chunks - 1, size)
  A[i:end_idx, , drop=FALSE] %% B
}
end <- Sys.time()

# Write results to file
sink("results.txt")
cat(sprintf("Cores: %d\nTime: %.2f seconds\nMatrix size: %d x %d\n\n",
            cores, as.numeric(end-start), size, size))
cat("== Matrix A ==\n")
print(A)
cat("\n== Matrix B ==\n")
print(B)
cat("\n== Result Matrix (A %% B) ==\n")
print(result)
sink()

stopCluster(cl)

Line: 1/35 Column: 1 Character: 108 (0x6C) Encoding: 1252 (ANSI - La
```



Pbs script to use for the above code is

```
#!/bin/bash

#PBS -N matrix_mult

#PBS -l nodes=1:ppn=20

#PBS -l walltime=00:05:00

#PBS -o matrix_mult_output.txt

Module load R

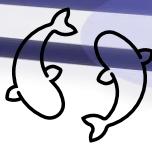
cd $PBS_O_WORKDIR

R CMD BATCH sample.R
```

Running the code and other processes are same, the output of the program is shown below.

```
test@master:~/singlecore
master:
      Req'd  Req'd   Elap
Job ID      Username Queue   Jobname   SessID NDS TSK Memory Time S Time
-----  -----
2371.master  test      workq  matrix_mu*   --    1 20    -- 00:05 F 00:00
(base) [test@master singlecore]$ cat results.txt
Cores: 20
Time: 0.09 seconds
Matrix size: 16 x 16

== Matrix A ==
 [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.5136103 0.86454708 0.331231904 0.39848139 0.11373490 0.6845677
[2,] 0.9173109 0.40882396 0.997196593 0.97064332 0.70002752 0.3838765
[3,] 0.1952504 0.01769765 0.773143439 0.27950195 0.87266719 0.3708466
[4,] 0.3486670 0.69085497 0.393423020 0.94911596 0.02615717 0.2956669
[5,] 0.9794336 0.84258851 0.254311273 0.92899903 0.74472009 0.4678345
[6,] 0.5914033 0.90072683 0.686426393 0.84594312 0.40871538 0.4820008
[7,] 0.5538387 0.02901857 0.664720071 0.21016974 0.12210918 0.1511475
[8,] 0.9264864 0.15588498 0.803234744 0.06015203 0.30623863 0.3372349
[9,] 0.7518538 0.49473596 0.571776467 0.72944155 0.78852570 0.3375359
[10,] 0.78771189 0.27391202 0.391756967 0.56395849 0.17692365 0.2304252
[11,] 0.5205900 0.18888903 0.822661406 0.49355026 0.40607348 0.4731600
[12,] 0.1007275 0.26424265 0.710678736 0.45908538 0.57566102 0.9248751
[13,] 0.8986169 0.81865533 0.980584489 0.34872321 0.37160966 0.2697134
[14,] 0.6916980 0.42478805 0.003245892 0.18736823 0.67167568 0.2414313
[15,] 0.2395366 0.82912238 0.545831241 0.89407115 0.22815947 0.2253813
[16,] 0.1967775 0.70687150 0.400159953 0.83418525 0.72704960 0.1085804
      [,7]      [,8]      [,9]      [,10]     [,11]     [,12]
[1,] 0.1313406 0.73470131 0.880992620 0.78142114 0.68393772 0.75634202
[2,] 0.6534323 0.02900606 0.310259715 0.54563301 0.69735596 0.04582394
[3,] 0.2684825 0.84366100 0.754390182 0.26346498 0.01299259 0.54492340
[4,] 0.4926363 0.48589895 0.260943690 0.50366830 0.73598529 0.33320558
[5,] 0.9501168 0.45437169 0.477298372 0.75240966 0.97354339 0.43447710
[6,] 0.4614451 0.30680546 0.795071260 0.58075656 0.74484603 0.64337628
[7,] 0.7880974 0.68211826 0.966176504 0.04366376 0.75088260 0.75863349
[8,] 0.1763962 0.40318433 0.708601501 0.20414223 0.40962954 0.91486345
[9,] 0.9853296 0.96345382 0.741893544 0.88756767 0.02257459 0.15550246
```



Transfer Files between Windows and HPC

You can transfer files from your HPC to your Windows machine by using the PuTTY's built-in SCP command (PSCP).

If PuTTY is installed, you can typically find *pscp.exe* in the same directory as PuTTY, such as *C:\Program Files\PuTTY* or *C:\Program Files(x86)\PuTTY*

Open command prompt and use the following command.

```
"C:\path\to\pscp.exe" username@hpc_address:/path/to/your/file.txt  
C:\destination\path
```

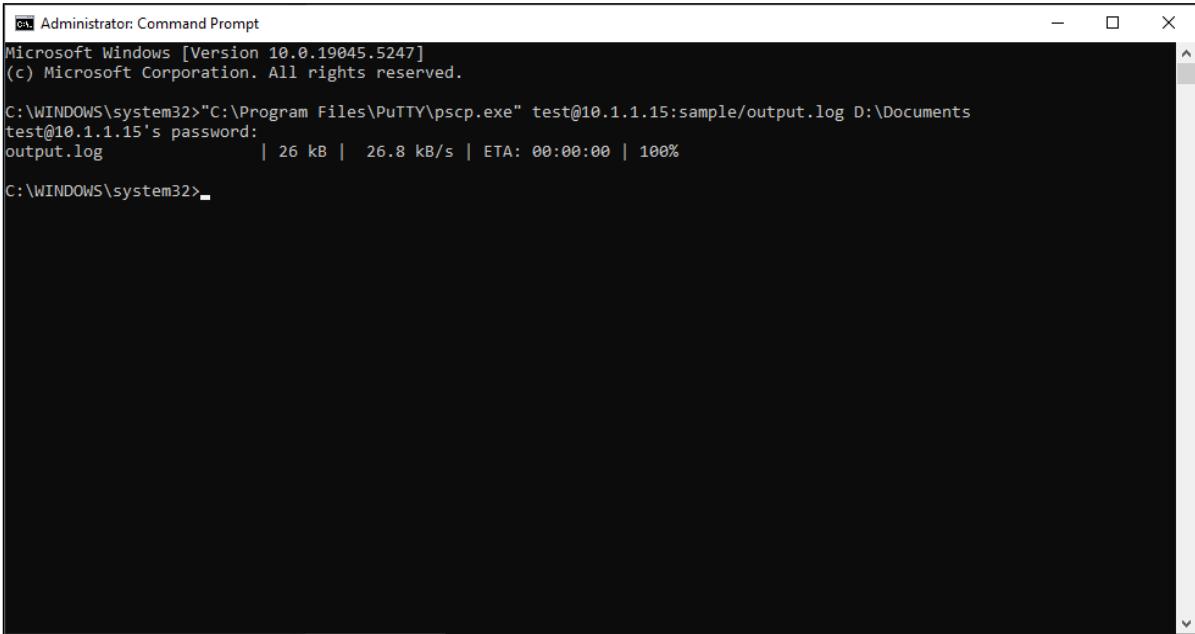
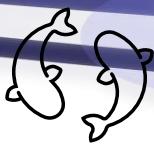
Replace:

- "C:\path\to\pscp.exe" with the full path to pscp.exe on your system.
- Username with your HPC username.
- hpc_address with the server address (e.g. 10.1.1.15).
- /path/to/your/file.txt with the location of the file on the HPC.
- C:\destination\path with the folder path on your Windows machine.

Example

```
"C:\Program Files\PuTTY\pscp.exe" test@10.1.1.15:sample/output.log  
D:\Documents
```

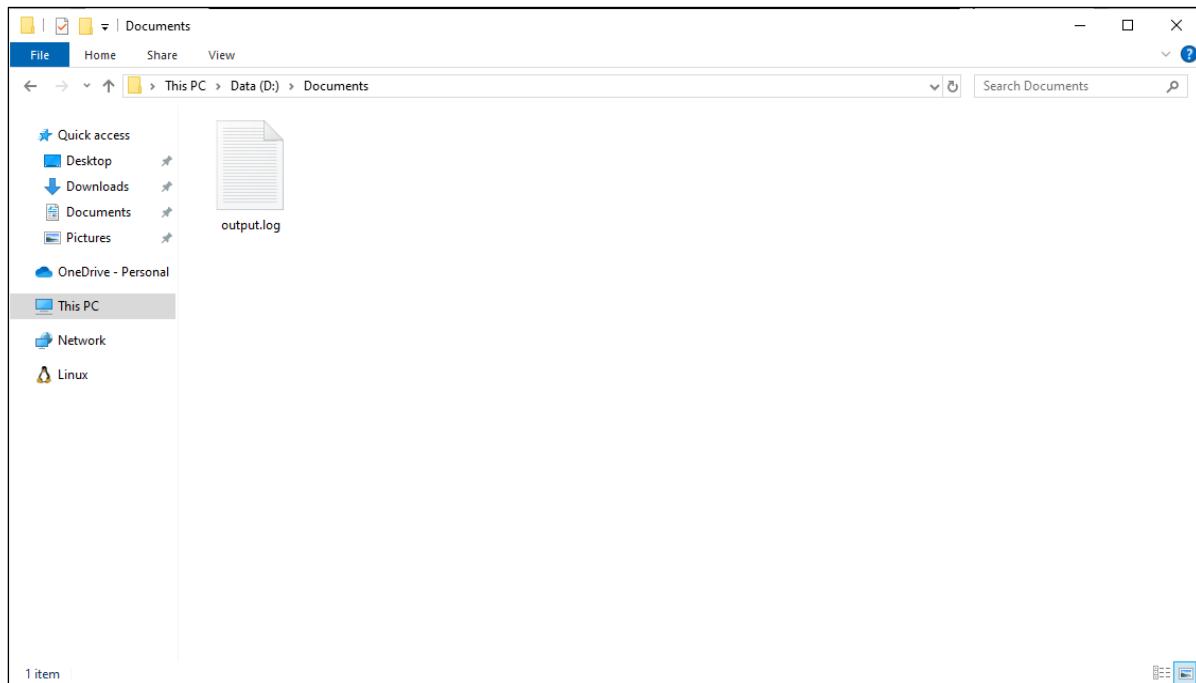
After executing the command, it will be prompted to enter your password for authentication. Once authenticated, the file will be downloaded to the specified directory. Ensure the file name is correctly typed, including capitalization (Linux paths are case-sensitive).



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

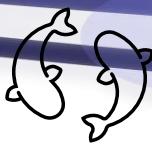
C:\WINDOWS\system32>"C:\Program Files\PuTTY\pscp.exe" test@10.1.1.15:sample/output.log D:\Documents
test@10.1.1.15's password:
output.log | 26 kB | 26.8 kB/s | ETA: 00:00:00 | 100%
C:\WINDOWS\system32>
```

**Wait for the file to download 100% before opening in your system.*

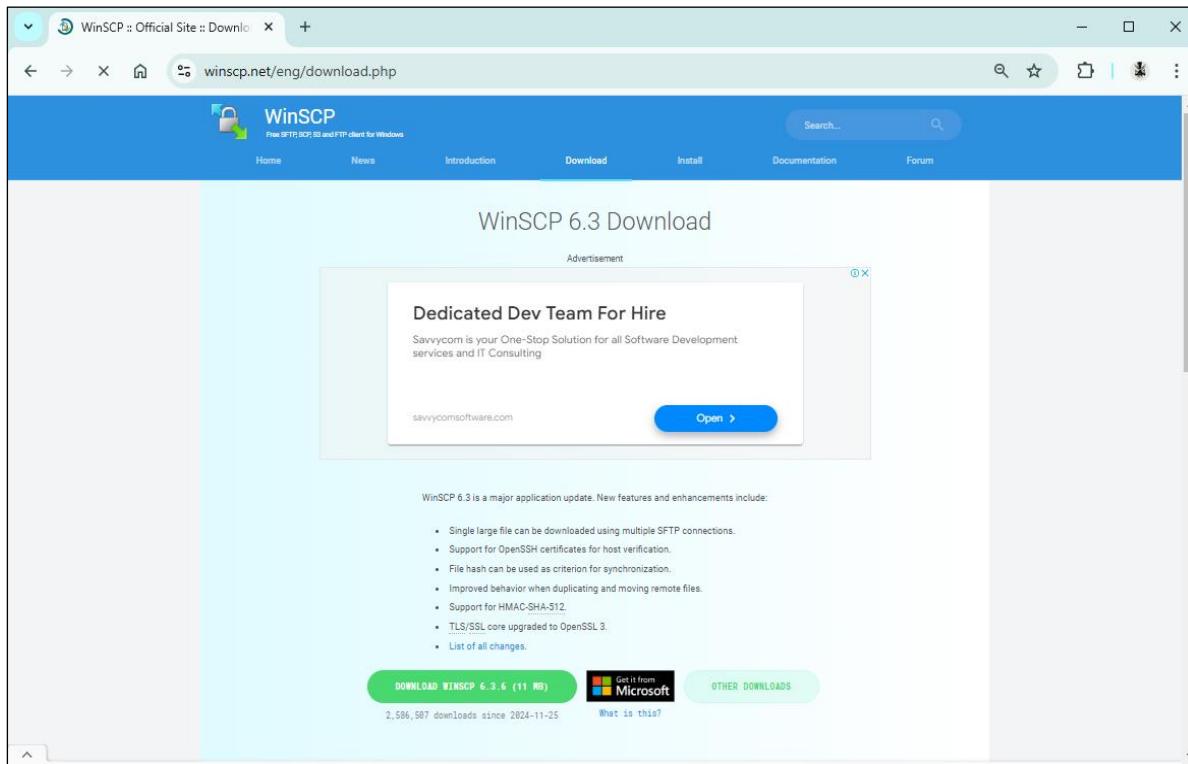


After downloading, you can see the files in your destination folder.

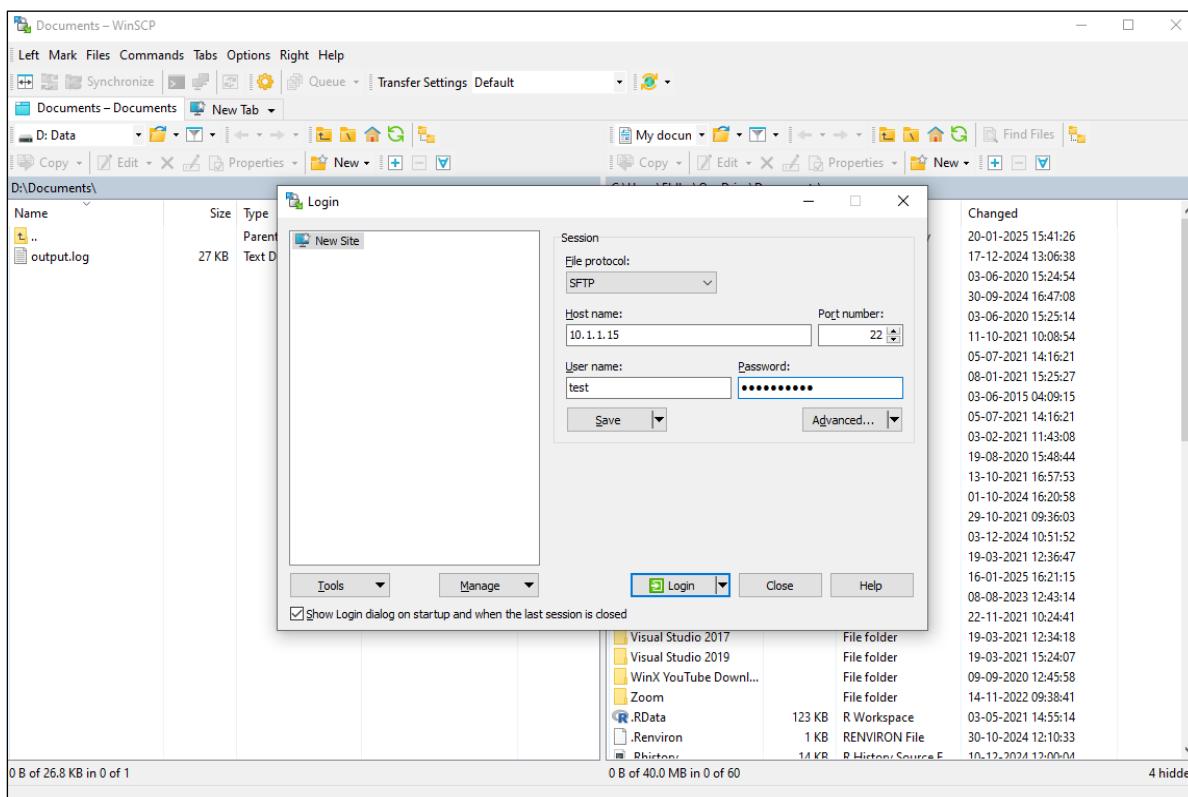
If you're not comfortable using the command line interface, you can use GUI-based software to transfer files between the two systems. Below are the instructions for using those tools.

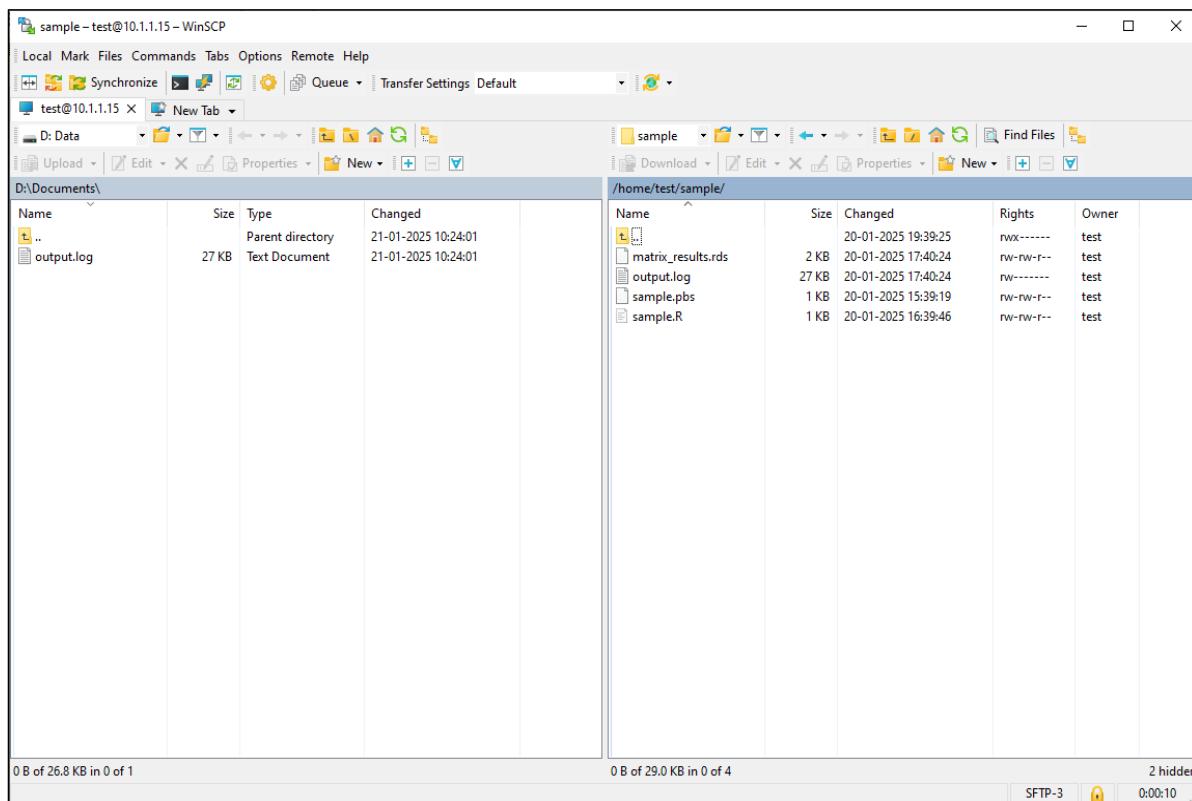
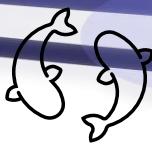


Various tools such as WinSCP and FileZilla, are available for transferring files between HPC and Windows systems. Below are some screenshots of the WinSCP software.

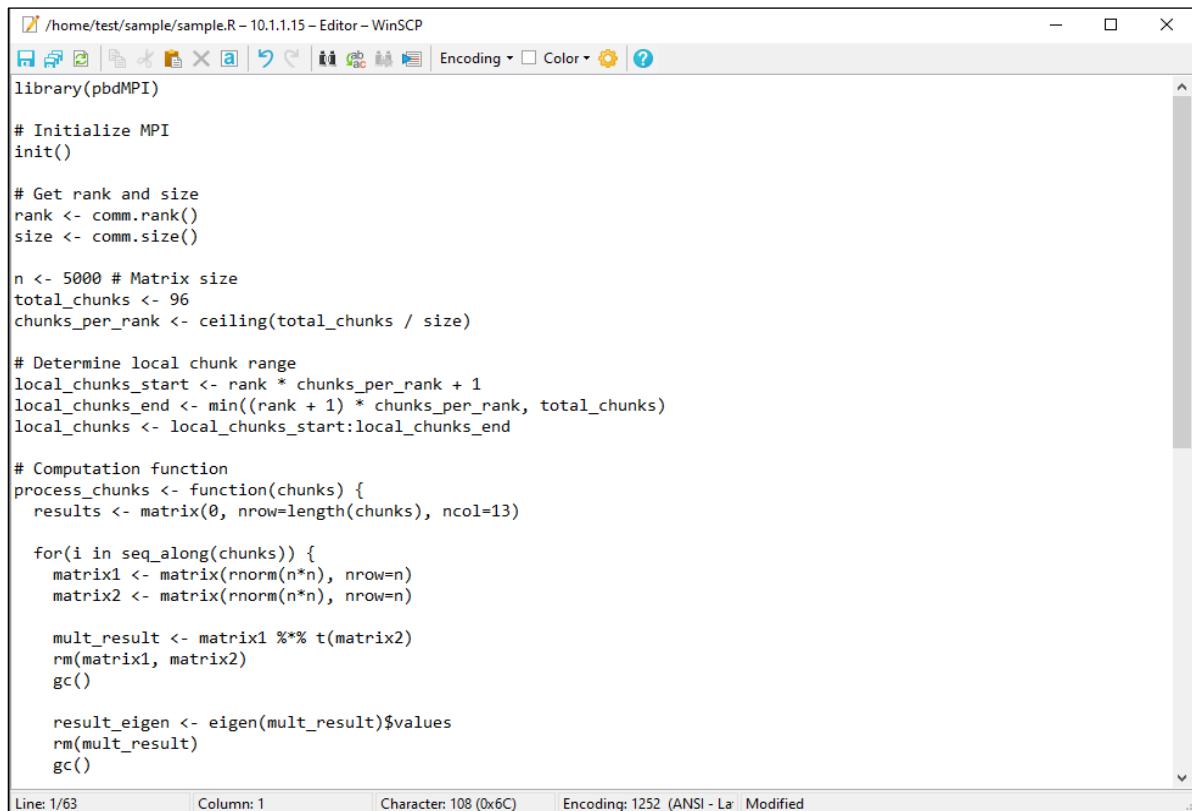


*Software download link is given in the Additional Resources section.





The left panel displays your Windows system files, while the right panel shows the HPC files. You can drag files between the panels or double-click to view or edit them.



```
library(pbdMPI)

# Initialize MPI
init()

# Get rank and size
rank <- comm.rank()
size <- comm.size()

n <- 5000 # Matrix size
total_chunks <- 96
chunks_per_rank <- ceiling(total_chunks / size)

# Determine local chunk range
local_chunks_start <- rank * chunks_per_rank + 1
local_chunks_end <- min((rank + 1) * chunks_per_rank, total_chunks)
local_chunks <- local_chunks_start:local_chunks_end

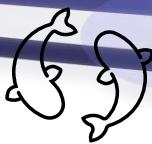
# Computation function
process_chunks <- function(chunks) {
  results <- matrix(0, nrow=length(chunks), ncol=13)

  for(i in seq_along(chunks)) {
    matrix1 <- matrix(rnorm(n*n), nrow=n)
    matrix2 <- matrix(rnorm(n*n), nrow=n)

    mult_result <- matrix1 %*% t(matrix2)
    rm(matrix1, matrix2)
    gc()

    result_eigen <- eigen(mult_result)$values
    rm(mult_result)
    gc()
  }
}
```

Line: 1/63 Column: 1 Character: 108 (0x6C) Encoding: 1252 (ANSI - La) Modified



Open R environment inside HPC

There is an alternate way to access R environment inside the HPC system. After successfully logging in:

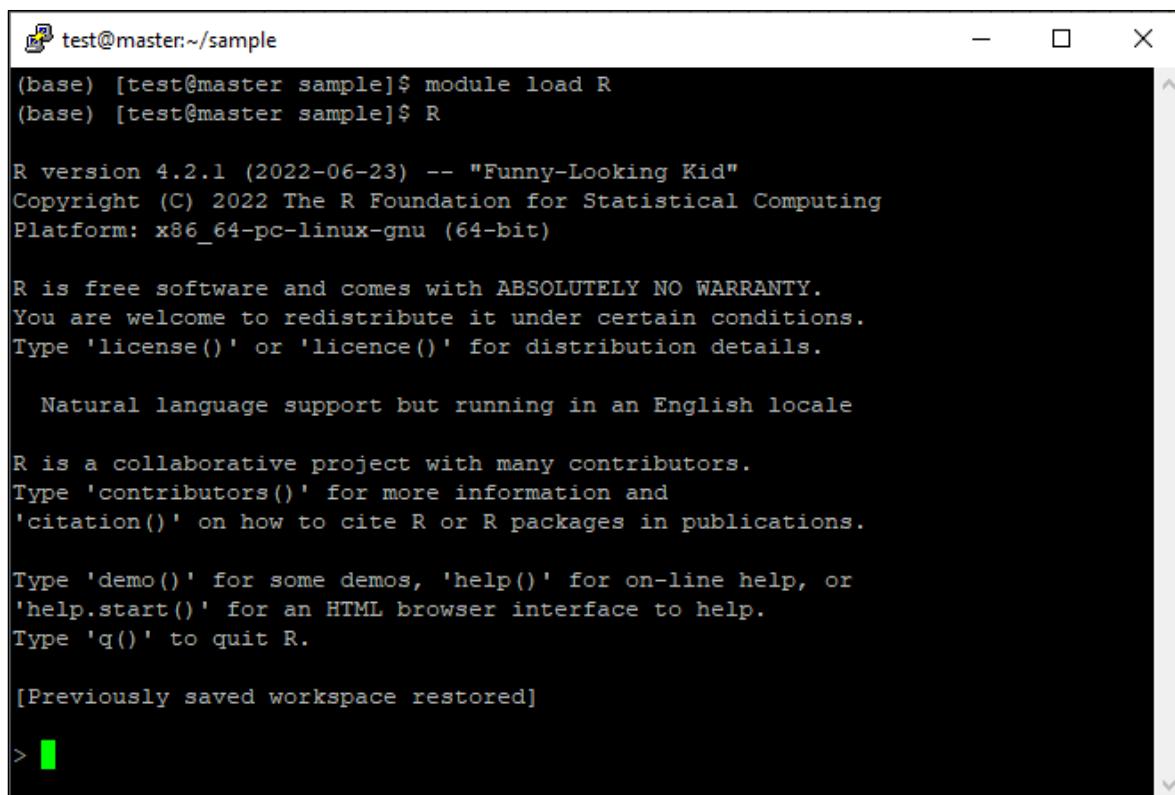
Start the R session by typing the following commands:

```
module load R
```

The first command loads the R module.

```
R
```

The second command starts the interactive R session. It will display the R version and related information shown in the picture below.



```
test@master:~/sample
(base) [test@master sample]$ module load R
(base) [test@master sample]$ R

R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

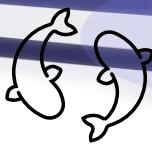
[Previously saved workspace restored]

> [green cursor]
```

To install a package

Before installing a package, you must set path to the shared storage, so that the package can be used by all the nodes in the hpc. To set path use this command.

```
.libPaths("/home/test/R/x86_64-redhat-linux-gnu-library/4.4")
```



After that you can install any package using the below command.

```
install.packages("package_name")
```

During the installation process, R will prompt you to select a CRAN mirror for downloading packages. You can choose either "42: India (Bengaluru)" or "43: India (Bhubaneswar)". To proceed, simply type **42** and press **Enter**.

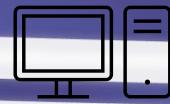
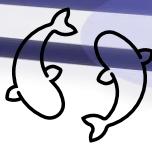
You can view the results saved from the *sample.R* script by loading the *.rds* file using R, follow these steps to see the results.

```
results <- readRDS("matrix_results.rds")

# View the first few rows of results
head(results)

# Summary statistics
summary(results)
```

```
test@master:~/sample
> results <- readRDS("matrix_results.rds")
> head(results)
   [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]
[1,] 2499.704 1445.551 0.41178855 500.7939 998.3236 1501.011 1994.895 2505.891
[2,] 2497.668 1441.420 1.58650653 498.0138 1007.7207 1503.833 2005.251 2506.610
[3,] 2498.881 1442.950 0.99865927 500.2015 1000.0142 1494.574 1991.885 2497.466
[4,] 2500.984 1444.410 0.07960888 500.8732 997.7694 1502.574 1986.487 2504.536
[5,] 2498.094 1444.072 1.03655203 499.7197 1000.3729 1504.579 1990.463 2487.215
[6,] 2499.946 1442.900 0.04602900 496.6904 997.8980 1501.634 1987.055 2513.232
   [,9]   [,10]  [,11]  [,12]  [,13]
[1,] 3001.955 3480.612 4004.056 4503.888 5114.587
[2,] 2986.397 3499.288 3998.191 4492.529 5089.636
[3,] 3002.595 3507.288 3995.097 4494.585 5096.468
[4,] 2987.952 3501.899 4004.510 4511.118 5096.978
[5,] 2989.564 3498.242 3995.310 4499.573 5181.135
[6,] 3002.158 3490.723 3996.313 4485.495 5079.488
> summary(results)
      V1        V2        V3        V4
Min. :2498  Min. :1441  Min. :0.001799  Min. :490.5
1st Qu.:2499  1st Qu.:1443  1st Qu.:0.164038  1st Qu.:497.2
Median :2500  Median :1444  Median :0.428425  Median :500.2
Mean   :2500  Mean   :1444  Mean   :0.555951  Mean   :499.8
3rd Qu.:2500  3rd Qu.:1444  3rd Qu.:0.934863  3rd Qu.:502.2
Max.   :2501  Max.   :1446  Max.   :2.094401  Max.   :508.1
      V5        V6        V7        V8        V9
Min. : 986.0  Min. :1486  Min. :1984  Min. :2479  Min. :2981
1st Qu.: 996.1  1st Qu.:1497  1st Qu.:1996  1st Qu.:2495  1st Qu.:2995
Median : 999.3  Median :1499  Median :2000  Median :2500  Median :3000
Mean   : 999.5  Mean   :1500  Mean   :1999  Mean   :2499  Mean   :3000
3rd Qu.:1003.2  3rd Qu.:1502  3rd Qu.:2003  3rd Qu.:2503  3rd Qu.:3005
Max.   :1011.4  Max.   :1516  Max.   :2015  Max.   :2517  Max.   :3019
      V10       V11       V12       V13
Min. :3478  Min. :3980  Min. :4481  Min. :5027
1st Qu.:3496  1st Qu.:3994  1st Qu.:4494  1st Qu.:5068
Median :3501  Median :3998  Median :4500  Median :5083
Mean   :3499  Mean   :3998  Mean   :4499  Mean   :5088
3rd Qu.:3505  3rd Qu.:4003  3rd Qu.:4504  3rd Qu.:5104
Max.   :3514  Max.   :4009  Max.   :4520  Max.   :5200
>
```



You can visualize the computed values using plots

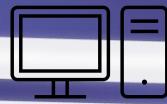
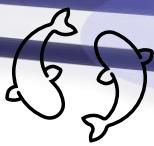
```
# Histogram of column V1
hist(results[, 1], main="Distribution of V1", xlab="Value", col="blue")

# Boxplot of standard deviation
boxplot(results[, 2], main="Standard Deviation of V2")

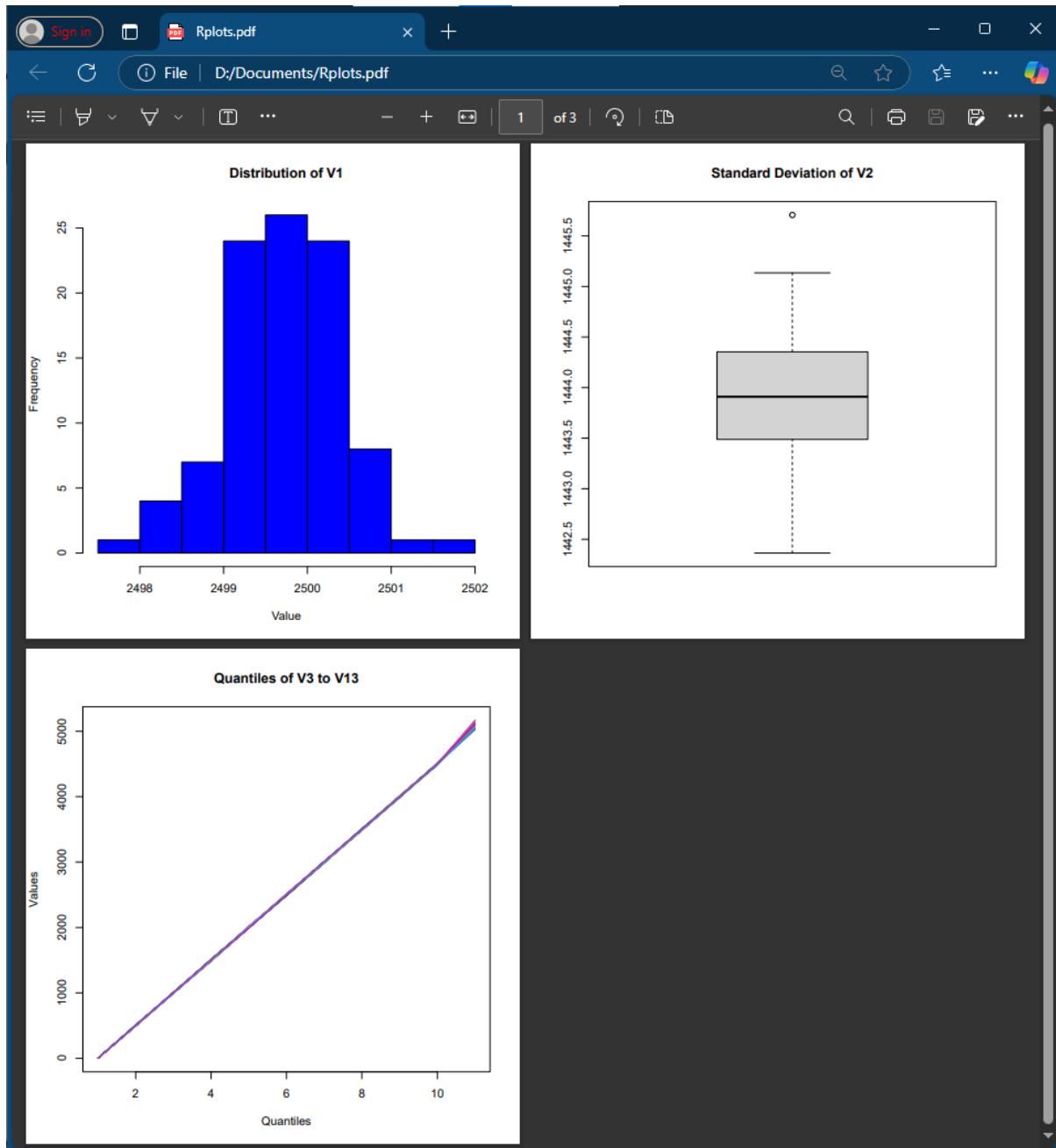
# Plot quantiles (columns 3 to 13)
matplot(t(results[, 3:13]), type="l", lty=1, main="Quantiles of V3 to V13",
xlab="Quantiles", ylab="Values")
```

After executing these commands, the console will not produce any output. Once you exit the R environment, a new file called **Rplots.pdf** will be generated in the current directory.

```
test@test:~/sample
> summary(results)
      V1          V2          V3          V4
Min. :2498  Min. :1442  Min. :0.002282  Min. :490.7
1st Qu.:2499  1st Qu.:1443  1st Qu.:0.105888  1st Qu.:497.5
Median :2500  Median :1444  Median :0.278905  Median :500.1
Mean   :2500  Mean   :1444  Mean   :0.452288  Mean   :500.2
3rd Qu.:2500  3rd Qu.:1444  3rd Qu.:0.643555  3rd Qu.:502.7
Max.  :2502  Max.  :1446  Max.  :2.069538  Max.  :512.4
      V5          V6          V7          V8          V9
Min. : 992.8  Min. :1483  Min. :1988  Min. :2485  Min. :2983
1st Qu.: 996.5  1st Qu.:1496  1st Qu.:1996  1st Qu.:2495  1st Qu.:2995
Median : 998.8  Median :1500  Median :2000  Median :2499  Median :2999
Mean   : 999.5  Mean   :1499  Mean   :2000  Mean   :2499  Mean   :2999
3rd Qu.:1001.7  3rd Qu.:1504  3rd Qu.:2004  3rd Qu.:2504  3rd Qu.:3003
Max.  :1012.5  Max.  :1518  Max.  :2020  Max.  :2516  Max.  :3012
      V10         V11         V12         V13
Min. : 3484  Min. :3984  Min. :4485  Min. :5027
1st Qu.: 3495  1st Qu.:3996  1st Qu.:4495  1st Qu.:5068
Median : 3499  Median :3999  Median :4499  Median :5082
Mean   : 3499  Mean   :3999  Mean   :4499  Mean   :5083
3rd Qu.: 3503  3rd Qu.:4003  3rd Qu.:4504  3rd Qu.:5098
Max.  : 3515  Max.  :4015  Max.  :4515  Max.  :5170
> hist(results[, 1], main="Distribution of V1", xlab="Value", col="blue")
> boxplot(results[, 2], main="Standard Deviation of V2")
> matplot(t(results[, 3:13]), type="l", lty=1, main="Quantiles of V3 to V13", xlab="Quantiles"
> q()
Save workspace image? [y/n/c]: y
(base) [test@test sample]$ ls
matrix_results.rds  output.log  Rplots.pdf  sample.pbs  sample.R
(base) [test@test sample]$
```



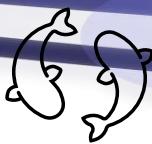
You can download the file to your windows system using any method and check the result.



Exiting the R Environment

To exit the R session when finished, simply type

```
q()
```



Good Practices for Using HPC

Request Resources Wisely: Only request the CPU, memory, and wall time your job needs to ensure efficient use of the system.

Avoid Login Node Overload: Do not run resource-intensive tasks on login nodes; always submit jobs through the scheduler.

Clean Up Regularly: Remove unused files and manage your storage to stay within quota limits.

Optimize Code Performance: Profile and optimize your code to improve efficiency, minimize bottlenecks, and reduce runtime.

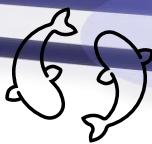
Monitor Jobs: Check your job logs and resource usage to optimize performance and identify issues.

Test Small, Scale Gradually: Run small test jobs before submitting large-scale jobs to estimate resource needs and debug issues.

Use Parallelization Effectively: Utilize parallel programming techniques (e.g., MPI, OpenMP, multi-threading) to make the best use of available cores and nodes.

Report Issues Promptly: Notify the HPC team if you encounter persistent issues, unexpected failures, or performance bottlenecks.

Follow Fair Use Policies: Use resources responsibly and share the HPC system fairly with other users.



FISH@CMFRI Power-Off Procedure

Stop PBS Jobs Gracefully

Log in to the head node and prevent new jobs from starting. You can do this by setting the nodes to an offline state.

```
qmgr -c "set server scheduling = false"
```

Check for running jobs and wait for them to finish, or you can choose to cancel the jobs.

After that set all nodes offline.

```
qmgr -c "set node compute-node[1-4] state=offline"
```

Shutdown Compute Nodes

Once jobs have stopped and nodes are offline, initiate the shutdown of compute nodes.

```
pdsh -w compute-node[1-4] "sudo shutdown now"
```

For all compute nodes, use

```
pdsh -a "sudo shutdown now"
```

Stop PBS Services

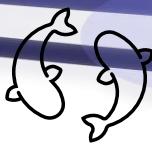
Stop the PBS server on the head node.

```
sudo systemctl stop pbs
```

Shutdown Head Node

After all services are stopped, shut down the head node.

```
sudo shutdown now
```



FISH@CMFRI Power-On Procedure

Power On Networking and Storage

Start by powering on the networking equipment (switches, routers), followed by storage systems like NAS or other shared storage devices.

Power On Head Node

Power on the head node and wait for it to boot fully.

Start PBS Services

Once the head node is online, start the PBS services.

```
sudo systemctl start pbs.service
```

Power On Compute Nodes

Power on the compute nodes using IPMI, a remote management tool, or manually.

Mount Shared File systems

On the Head node, ensure the shared file systems are mounted.

```
sudo mount -a
```

Bring Compute Nodes Online In PBS

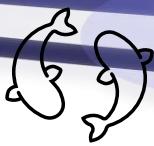
Set the compute nodes back to an online state in PBS.

```
qmgr -c "set node compute-node[1-4] state=free"
```

Enable Job Scheduling

Re-enable job scheduling once everything is operational.

```
qmjc -c "set server scheduling = true"
```



Verify Cluster Health

Verify that the PBS services are running properly and all nodes are online.

```
pbsnodes -a
```

Check if any nodes are in an offline state or have errors and correct them.

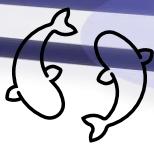
Verify Compute Node NFS

Verify that all NFS shares are mounted properly before executing jobs,

```
pdsh -w compute-node[1-4] df -h
```

If all three NFS shares are mounted proceed to run jobs, else mount those shares with below command,

```
pdsh -w compute-node[1-4] mount -a
```



Installing Packages in Compute Node Image

Step 1: Export the image path as shown below

```
#export CHROOT=/opt/ohpc/admin/images/rocky8.8
```

Step 2: Install the required package in master node and install the same in image using below command.

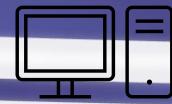
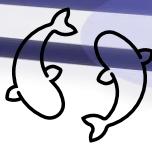
```
#dnf -y --installroot $CHROOT install <package-name>
```

Step 3: Run the warewulf virtual node file system command

```
#wwvnfs --chroot $CHROOT
```

Step 4: Reboot the compute nodes to start using the installed package.

```
#pdsh -w compute-node[1-4] reboot
```



Appendix

C code used for matrix multiplication

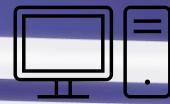
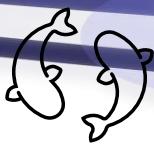
```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define N 16 // Matrix size (N x N)

// Function to initialize a matrix with random values
void initialize_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = rand() % 10; // Random values between 0 and 9
        }
    }
}

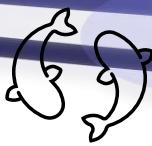
// Function to print a matrix
void print_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%.2f ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main(int argc, char *argv[]) {
    int rank, size;
    double A[N][N], B[N][N], C[N][N]; // Matrices A, B, and C
    double local_A[N][N], local_C[N][N]; // Local parts of A and C
    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```



```
MPI_Comm_size(MPI_COMM_WORLD, &size);
if (N % size != 0) {
    if (rank == 0) {
        fprintf(stderr, "Matrix size N must be divisible by number of processes.\n");
    }
    MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
}

int rows_per_process = N / size; // Number of rows per process
// Initialize matrices on rank 0
if (rank == 0) {
    initialize_matrix(A);
    initialize_matrix(B);
    printf("Matrix A:\n");
    print_matrix(A);
    printf("\nMatrix B:\n");
    print_matrix(B);
}
// Broadcast matrix B to all processes
MPI_Bcast(B, N * N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// Scatter rows of matrix A to all processes
MPI_Scatter(A, rows_per_process * N, MPI_DOUBLE, local_A, rows_per_process *
            N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// Start timing the computation
double start_time = MPI_Wtime();
// Compute the local part of the result matrix C
for (int i = 0; i < rows_per_process; i++) {
    for (int j = 0; j < N; j++) {
        local_C[i][j] = 0.0;
        for (int k = 0; k < N; k++) {
            local_C[i][j] += local_A[i][k] * B[k][j];
        }
    }
}
```



```
}

// Gather the local results into the result matrix C

MPI_Gather(local_C, rows_per_process * N, MPI_DOUBLE, C, rows_per_process * N,
           MPI_DOUBLE, 0, MPI_COMM_WORLD);

// End timing the computation

double end_time = MPI_Wtime();

// Print the result matrix on rank 0

if (rank == 0) {

    printf("\nResult Matrix C:\n");

    print_matrix(C);

    // Print the time taken for the computation

    printf("\nTime taken: %.6f seconds\n", end_time - start_time);

}

MPI_Finalize();

return EXIT_SUCCESS;

}
```

R code used for parallel matrix operations

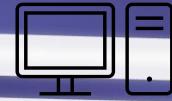
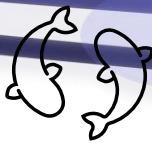
```
library(pbdMPI)

# Initialize MPI
init()

# Get rank and size
rank <- comm.rank()
size <- comm.size()

n <- 5000 # Matrix size
total_chunks <- 96
chunks_per_rank <- ceiling(total_chunks / size)

# Determine local chunk range
```



```
local_chunks_start <- rank * chunks_per_rank + 1
local_chunks_end <- min((rank + 1) * chunks_per_rank, total_chunks)
local_chunks <- local_chunks_start:local_chunks_end

# Computation function
process_chunks <- function(chunks) {
  results <- matrix(0, nrow=length(chunks), ncol=13)

  for(i in seq_along(chunks)) {
    matrix1 <- matrix(rnorm(n*n), nrow=n)
    matrix2 <- matrix(rnorm(n*n), nrow=n)

    mult_result <- matrix1 %*% t(matrix2)
    rm(matrix1, matrix2)
    gc()

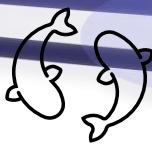
    result_eigen <- eigen(mult_result)$values
    rm(mult_result)
    gc()

    results[i,] <- c(
      mean(abs(result_eigen)),
      sd(abs(result_eigen)),
      quantile(abs(result_eigen), probs=seq(0,1,0.1))
    )
  }

  return(results)
}

# Start timer
if(rank == 0) start_time <- Sys.time()

# Compute local results
```



```
local_results <- process_chunks(local_chunks)

# Gather results from all ranks
all_results <- gather(local_results)

# Have rank 0 combine and save results
if(rank == 0) {
  final_results <- do.call(rbind, all_results)
  end_time <- Sys.time()
  print(paste("Time taken:", end_time - start_time))
  saveRDS(final_results, "matrix_results.rds")
}

# Finalize MPI
finalize()
```

R code used for parallel matrix multiplication on single node

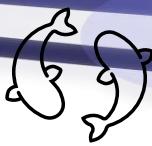
```
library(parallel)
library(doParallel)

size <- 16
cores <- 20

# Setup local cluster
cl <- makeCluster(cores)
registerDoParallel(cl)

# Create matrices
A <- matrix(runif(size^2), size)
B <- matrix(runif(size^2), size)

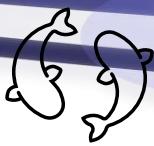
start <- Sys.time()
chunks <- ceiling(size/cores)
```



```
result <- foreach(i = seq(1, size, by=chunks), .combine='rbind') %dopar% {  
  end_idx <- min(i + chunks - 1, size)  
  A[i:end_idx, , drop=FALSE] %*% B  
}  
end <- Sys.time()  
# Write results to file  
sink("results.txt")  
cat(sprintf("Cores: %d\nTime: %.2f seconds\nMatrix size: %d x %d\n\n",  
           cores, as.numeric(end-start), size, size))  
cat("== Matrix A ==\n")  
print(A)  
cat("\n== Matrix B ==\n")  
print(B)  
cat("\n== Result Matrix (A %*% B) ==\n")  
print(result)  
sink()  
stopCluster(cl)
```

Job Status Codes and Their Descriptions

Status Letter	Meaning	Description
Q	Queued	The job is waiting in the queue to be scheduled for execution.
R	Running	The job is currently running on the assigned resources.
E	Exiting	The job has finished running and is performing clean-up tasks.
C	Completed	The job has finished execution successfully or has been terminated.
H	Held	The job is held and will not be scheduled for execution until it is released.
W	Waiting	The job is waiting for a specific condition to be met (e.g., a start time).
T	Transit	The job is being moved to another server or is in the process of being staged.
F	Finished	The job has completed its lifecycle (may have finished successfully or failed).



Additional Resources

HPC – High Performance Computing

<https://cloud.google.com/discover/what-is-high-performance-computing?hl=en>

<https://www.ibm.com/think/topics/hpc>

PBS Professional User's Guide

<https://help.altair.com/2024.1.0/PBS Professional/PBSUserGuide2024.1.pdf>

PBS Professional Administrator's Guide

<https://help.altair.com/2024.1.0/PBS Professional/PBSAdminGuide2024.1.pdf>

Linux Tutorial

<https://www.tutorialspoint.com/unix/index.htm>

R Tutorial

<https://www.w3schools.com/r>

C Tutorial

<https://www.w3schools.com/c>

Introduction to parallel computing with R

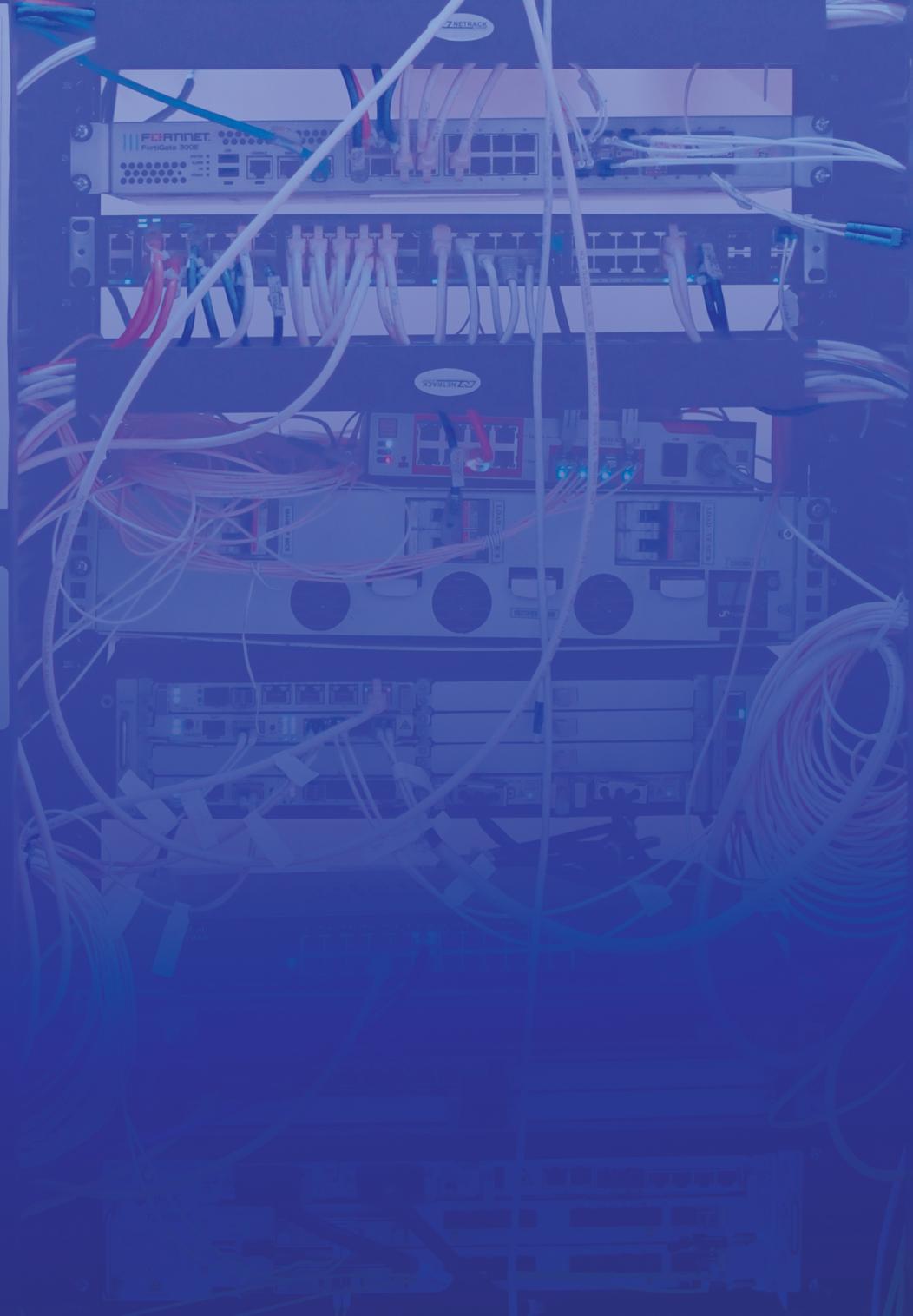
<https://rawgit.com/PPgp/useR2017public/master/tutorial.html>

A Quick Guide for the pbdMPI Package

<https://cran.r-project.org/web/packages/pbdMPI/vignettes/pbdMPI-guide.pdf>

WinSCP Software

<https://winscp.net/eng/download.php>



ICAR-Central Marine Fisheries Research Institute

(Department of Agricultural Research and Education, Government of India)

Department of Agricultural Research and Education, Government of India
P.B. No. 1603, Ernakulam North P.O., Kochi - 682 018, Kerala, India

