



PARAM Ganga

User's Manual

Ver. 1.0

Last updated: April 22, 2022

www.cdac.in

Copyright Notice

Copyright © 2022 Centre for Development of Advanced Computing
All Rights Reserved.

Any technical documentation that is made available by C-DAC (Centre for Development of Advanced Computing) is the copyrighted work of C-DAC and is owned by C-DAC. This technical documentation is being delivered to you as is, and C-DAC makes no warranty as to its accuracy or use. Any use of the technical documentation or the information contained therein is at the risk of the user. C-DAC reserves the right to make changes without prior notice.

No part of this publication may be copied without the express written permission of C-DAC.

Trademarks

CDAC, CDAC logo, NSM logo are trademarks or registered trademarks.

Other brands and product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.



Intended Audience

This document is meant for PARAM Ganga users.

Typographic Conventions

Symbol	Meaning
Blue underlined text	A hyperlink or link you can click to go to a related section in this document or to a URL in your web browser.
Bold	The names of menus, menu items, headings, and buttons.
<i>Italics</i>	Variables or placeholders or special terms in the document.
<code>Console text</code>	Console commands



Getting help

For technical assistance or license renewal, please send an email to gangasupport@iitr.ac.in.

Give us your feedback

We value your feedback. Kindly send your comments on the content of this document to gangasupport@iitr.ac.in. Please include the page number of the document along with your feedback.



DISCLAIMER

The information contained in this document is subject to change without notice. C-DAC shall not be liable for errors contained herein or for incidental or consequential damages in connection with the performance or use of this manual.

Contents

Introduction.....	9
System Architecture and Configuration.....	10
System Hardware Specifications	10
Master Nodes	10
Login Nodes	11
Service Nodes	11
CPU Compute Nodes	11
GPU Compute Nodes.....	12
Storage	12
Operating System	13
PARAM Ganga Architecture Diagram.....	13
Primary Interconnection Network	14
Secondary Interconnection Network	14
Software Stack.....	14
First Things First.....	18
Getting an Account on PARAM Ganga	18
First login	18
Forgot Password?	19
System Access	20
Remote Access	20
Transferring files between local machine and HPC cluster.....	21
Tools	23
Running Interactive Jobs.....	25
Managing Jobs through its Lifecycle	26
walltime	26
List Partition	27
Addressing Basic Security Concerns	33

More about Batch Jobs (SLURM)	34
Parameters used in SLURM job script	34
I am familiar with PBS/ TORQUE. How do I migrate to SLURM?	37
Preparing Your Own Executable	39
Spack	43
Introduction.....	43
To Use Pre-Installed Applications from Spack	44
To install new application.....	45
Uninstalling Packages	47
Using Environments	47
Packaging (For Application developers).....	48
Sample SLURM script for OpenMP applications/programs. to use Spack.....	50
Sample SLURM script for MPI applications/programs to use Spack.....	50
Job Scheduling on PARAM Ganga	52
Scheduler.....	52
sinfo	53
PARAM Ganga SLURM Partitions and QoS.....	53
For Submitting the job.....	54
walltime	55
Debugging Your Codes	59
Introduction.....	59
Basics How Tos	59
Conclusions.....	80
Points to Note.....	80
Overall Coding Modifications Done	81
Machine Learning / Deep Learning Application Development	82
How to Install your own Software?	83
Some Important Facts	85
About File Size	85

Little-Endian and Big-Endian issues?	86
Best Practices for HPC	87
Installed Applications/Libraries	88
Standard Application Programs on PARAM Ganga	88
LAMMPS Applications	89
GROMACS APPLICATION	91
Acknowledging the National Supercomputing Mission in Publications.....	94
Getting Help – PARAM Ganga Support.....	95
Steps to Create a New Ticket	95
Closing Your Account on PARAM Ganga	97
References.....	104

List of Figures

Figure 1 - PARAM Ganga Architecture Diagram	13
Figure 2 – Software Stack.....	16
Figure 3 - A snapshot of command using MobaXterm	23
Figure 4 - A snapshot of "scp" tool to transfer file to and from remote computer.	24
Figure 5 – Enter Captcha/String.....	24
Figure 6 - Output of sinfo command.....	27
Figure 7 – Snapshot depicting the usage of “Job Array”	29
Figure 8 – scontrol show node displays compute node information	30
Figure 9 – scontrol show partition displays specific partition details	30
Figure 10 – scontrol show job displays specific job information.....	31
Figure 11 – sinfo Command	53
Figure 12 - Listing the shares of association to a cluster.....	57
Figure 13 – Snapshot of debugging process	62
Figure 14 – Snapshot of debugging process	63
Figure 15- Output at a debugging stage	64
Figure 16 – Snapshot of debugging process	65
Figure 17 – Output depicting “Arithmetic Exception”	66
Figure 18 – Snapshot of debugging process	66
Figure 19 – Well, we dumped core!!	66
Figure 20 - Snapshot of debugging process.....	67
Figure 21 – Setting Breakpoint	68
Figure 22 – single stepping through to catch error!!.....	69
Figure 23 – Debugging continued	70
Figure 24 – Debugging continued	70
Figure 25 – Setting a watch point	71
Figure 26 – Debugging continued.....	72
Figure 27 – Well, back to square one!!.....	73
Figure 28 – Again Dumping Core!! Things are getting interesting or frustrating or both !! ...	74
Figure 29 – Debugging continued.....	74
Figure 30 – Debugging continued.....	75
Figure 31 – Debugging continued (Will it ever end?)	76
Figure 32 – We are almost there!!.....	76
Figure 33 – Debugging continued	77
Figure 34 – At last a clue!!!	78
Figure 35 - Correction applied!!.....	79

Figure 36 – Resolved!!!	80
Figure 37 – What all we did to get things right!	81
Figure 38 – Snapshot of Ticketing System	95
Figure 39 - Snapshot of Ticketing System	96
Figure 40 - Snapshot of Ticketing System	96
Figure 41 - Snapshot of Ticketing System	97

Introduction

This document is the user manual for the PARAM Ganga Supercomputing facility at IIT Roorkee. It covers a wide range of topics ranging from a detailed description of the hardware infrastructure to the information required to utilize the supercomputer, such as information about logging on to the supercomputer, submitting jobs, retrieving the results on to user's Laptop/ Desktop etc. In short, the manual describes all that one needs know to effectively utilize PARAM Ganga.

The supercomputer PARAM Ganga is based on a heterogeneous and hybrid configuration of Intel Xeon Cascade lake processors, and NVIDIA Tesla V100. The system was designed and implemented by HPC Technologies team, Centre for Development of Advanced Computing (C-DAC).

It consists of 2 Master nodes, 10 Login nodes, 8 Service/Management nodes and 312 (CPU+GPU+HM) nodes with total peak computing capacity of 1.67 (CPU+GPU+HM) PFLOPS performance.

System Architecture and Configuration

System Hardware Specifications

PARAM Ganga system is based on processor Intel Xeon Platinum 8268, NVIDIA Tesla V100 with total peak performance of 1.6 PFLOPS. The cluster consists of compute nodes connected with Mellanox(HRD) InfiniBand interconnect network. The system uses the Lustre parallel file system.

- Total number of nodes: 332(20+312)
 - Service nodes: 20**(Master+ Login +Service +Management Nodes)
 - CPU only nodes: 150
 - GPU ready nodes: 64
 - GPU nodes: 20
 - High Memory nodes:39

Master Nodes

PARAM Ganga is an aggregation of a large number of computers connected through networks. The basic purpose of the master node is to manage and monitor each of the constituent components of PARAM Ganga from a system’s perspective. This involves operations like monitoring the health of the components, the load on the components, the utilization of various sub-components of the computers in PARAM Ganga.

Master Nodes: 2	
2* Intel Xeon G-6248	Total Cores = 80 cores
Cores =40, 2.5 GHz	
Memory= 384 GB	Total Memory = 768 GB
HDD = 1 TBx8	

Login Nodes

Login nodes are typically used for administrative tasks such as editing, writing scripts, transferring files, managing your jobs and the like. You will always get connected to one of the login nodes. From the login nodes you can get connected to a compute node and execute an interactive job or submit batch jobs through the batch system (SLURM) to run your jobs on compute nodes. For ALL users PARAM Ganga login nodes are the entry points and hence are shared. By default, there will be a limit on the CPU time that can be used on a login node by a user and there is a limit/user on the memory as well. If any of these are exceeded, the job will get terminated.

Login Nodes: 10	
2* Intel Xeon G-6248	Total Cores = 800 cores
Cores = 40, 2.5 GHz	
Memory= 384 GB	Total Memory = 3072 GB
HDD = 1 TBx8	

Service Nodes

Typically, the purpose of the service node is to provide Security, Management, Monitoring and other services to the cluster.

Service Nodes: 8	
2* Intel Xeon G-6248	Total Cores = 800 cores
Cores = 40, 2.5 GHz	
Memory= 384 GB	Total Memory= 3072 GB
HDD = 1 TBx5	

CPU Compute Nodes

CPU nodes are indeed the work horses of PARAM Ganga. All the CPU intensive activities are carried on these nodes. Users can access these nodes from the login node to run interactive or batch jobs. Some of the nodes have higher memory, which can be exploited by users in the aforementioned way.

CPU only Compute Nodes: 150	
2* Intel Xeon Platinum 8268	Total Cores = 7200 cores
Cores = 48, 2.9 GHz	
Memory= 192 GB, DDR4 2933 MHz	Total Memory=28800 GB

 SSD = 480 GB

GPU ready Compute Nodes: 64

2* Intel Xeon Platinum 8268

Cores = 48, 2.9 GHz

Memory= 192 GB, DDR4 2933 MHz

SSD = 480 GB

Total Cores = 3072 cores

Total Memory= 12,288 GB

GPU Compute Nodes

GPU compute nodes are the nodes that have CPU cores along with accelerators cards. For some applications GPUs get markedly high performance. For exploiting these, one has to make use of special libraries which map computations on the Graphical Processing Units (Typically one has to make use of CUDA or OpenCL).

GPU Compute Nodes: 20

2* Intel Xeon Gold-6248

Cores = 40, 2.5 GHz

Memory= 192 GB, DDR4 2933 MHz

SSD = 480 GB (local scratch) per node

2*Nvidia V100 per node

GPU Cores per node= 2*5120= 10240

GPU Memory = 16 GB HBM2 per Nvidia V100

Total Cores = 800 cores

Total Memory= 3840 GB

CPU only Compute Nodes with High memory: 78

2* Intel Xeon Platinum 8268

Cores = 48, 2.9 GHz

Memory= 768 GB, DDR4 2933 MHz

SSD = 480 GB

Total Cores = 3744 cores

Total Memory=59904 GB

Storage

- Based on Lustre parallel file system
- Total useable capacity of 2.2PB Primary storage

- Throughput 50 GB/s

Operating System

- Operating system on PARAM Ganga is Linux – CentOS 7.9

PARAM Ganga Architecture Diagram

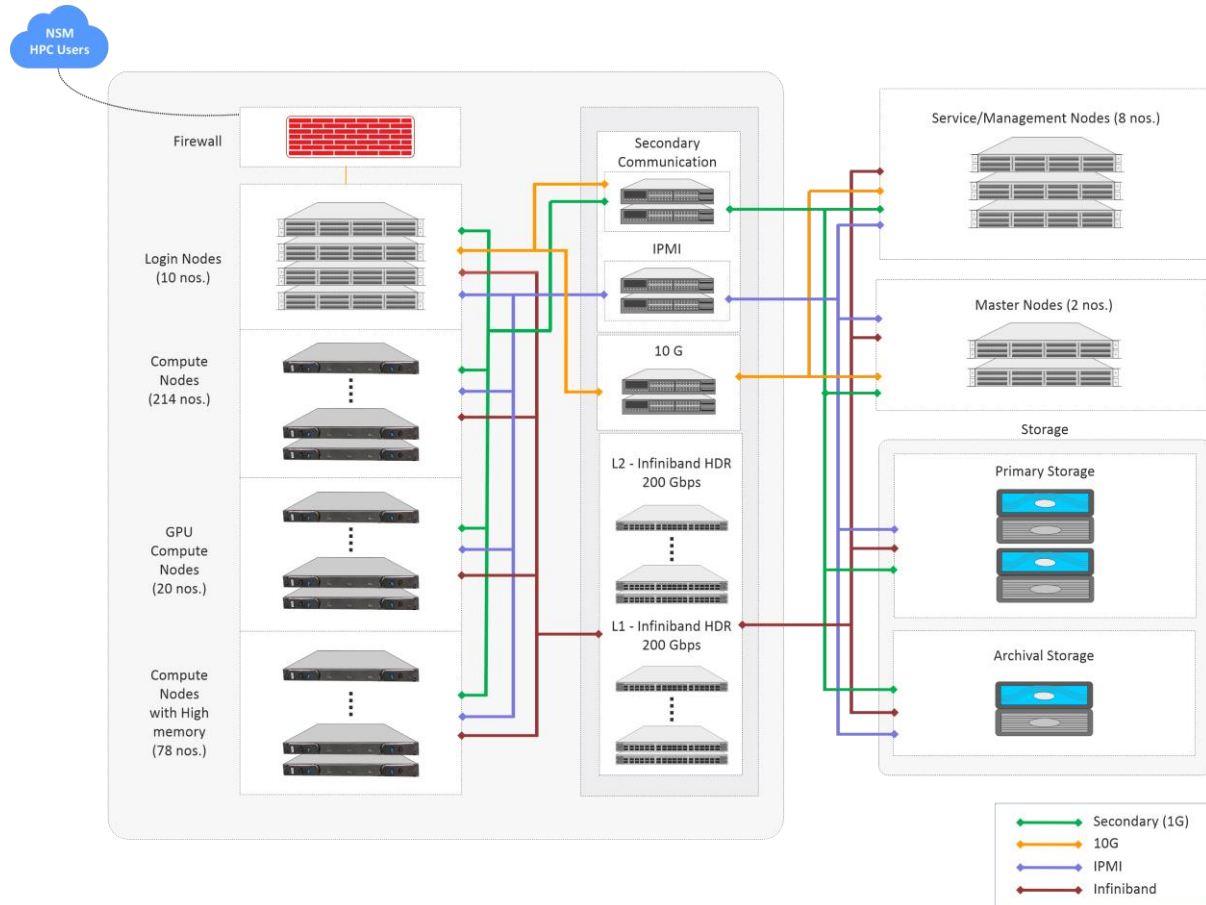


Figure 1 - PARAM Ganga Architecture Diagram

Network infrastructure

A robust network infrastructure is essential to implement the basic functionalities of a cluster. These functionalities are:

- Management functionalities i.e., to monitor, troubleshoot, start, stop various components of the cluster, etc. (Network/portion of Network which implements this functionality is referred to as Management fabric).
- Ensuring fast read/write access to the storage (Network/portion of Network which implements this functionality is referred to as storage fabric).

- c) Ensuring fast I/O operations like connecting to other clusters, connecting the cluster to various users on the campus LAN, etc. (Network/portion of Network which implements this functionality is referred to as I/O Fabric).
- d) Ensuring high-bandwidth, low-latency communication amongst processors to for achieving high-scalability (Network/portion of Network which implements this functionality is referred to as Message Passing Fabric)

Technically, all the aforementioned functionalities can be implemented in a single network. From the perspectives of requirements, optimal performance and economic suitability, the aforementioned functionalists are implemented using two different networks based on different technologies, as mentioned next:

Primary Interconnection Network

Computing nodes of PARAM Ganga are interconnected by a high-bandwidth, low-latency interconnect network.

InfiniBand: HDR 100 Gbps

InfiniBand is a high-performance communication architecture owned by Mellanox. This communication architecture offers low communication latency, low power consumption and a high throughput. All CPU nodes are connected via the InfiniBand interconnect network.

Secondary Interconnection Network

Gigabit Ethernet: 10 Gbps

Gigabit Ethernet is the interconnection network that is most commonly available. For Gigabit Ethernet, no additional modules or libraries are required. The Open MPI, MPICH implementations will work over Gigabit Ethernet.

Software Stack

Software Stack is an aggregation of software components that work in tandem to accomplish a given task. The task can be, to facilitate a user to execute his job/s or to facilitate a system administrator to manage a system efficiently. In effect, the software will have all the necessary components to accomplish a given task. There may be multiple components of different flavors to accomplish a given sub-task. The user/administrator may mix and match these components depending on his choice. Typically, a user would be

interested in preparing his executable, executing the same with his data sets and visualize the output generated by him. For accomplishing the same, the user would need to compile his codes, link the codes with communication libraries, math libraries, numerical algorithm libraries, prepare the executable, run the same with desired data sets, monitor the progress of his jobs, gathering the results and visualizing the output.

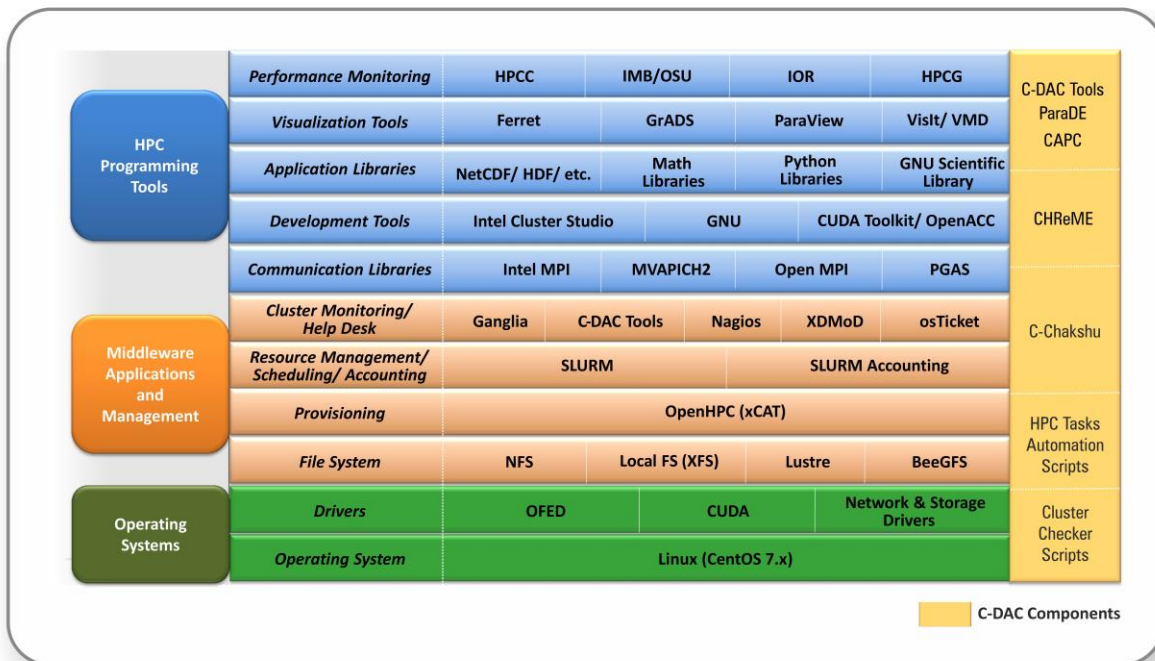
Typically, a system administrator would be interested in ensuring that all the resources are optimally utilized. For accomplishing this, he may need some installation tools, tools for checking the health of all the components, good schedulers, tools to facilitate allocation of resources to users and monitor the usage of the resources.

The software stack provided with this system has a gamut of software components that meets all the requirements of a user and that of a system administrator. The components of the software stack are depicted in figure 2.

Amongst these, C-CHAKSHU has been recently developed and deployed by CDAC. We solicit your feedback on these tools at gangasupport@iitr.ac.in.

C-CHAKSHU: This is a multi-cluster management tool that facilitates the administrator to efficiently operate the HPC facility. It also enables the user to monitor system metrics relating to CPU, Storage, Interconnects, File system and Application specific utilization from a single dashboard. For more information, please follow the link. <http://paramganga.iitr.ac.in:4200/chakshu-front>

System Software Stack



C-DAC HPC System Software Stack

Figure 2 – Software Stack

Functional Areas	Components
Base OS	CentOS 7.9
Architecture	x86_64
Provisioning	xCAT 2.16.3
Cluster Manager	Openhpc (1.3.8)
Monitoring Tools	C-CHAKSHU, Nagios, Ganglia, XDMoD

Resource Manager	Slurm-20.11.8
I/O Services	Lustre Client-x.xx.x
High Speed Interconnects	Mellanox InfiniBand
Compiler Families	GNU (gcc, g++, gfortran) Intel Compiler (icc, ifort, icpc)
MPI Families	MVAPICH, OpenMPI, MPICH

First Things First

Getting an Account on PARAM Ganga

To begin with, you need to get an account on PARAM Ganga. This is a very easy process. Please follow the steps given below:

- a) Download the ‘User Account Creation Form’ by following the link <https://paramganga.iitr.ac.in/ucform>
- b) Fill the relevant details.
- c) Get the signatures of your Head of the Department and the ‘approving authority’.

Note:

- For IIT Roorkee internal user, users will have the approving authority from IIT Roorkee. They can submit it to the “PARAM Ganga System Administrator* or email a scanned copy to. gangasupport@iitr.ac.in.
 - For Users who are not from IIT Roorkee will have to email the scanned copy to gangasupport@iitr.ac.in, HoD HPC-Tech CDAC will be the approving authority for them.
- d) You will receive an email in your official email ID intimating the creation of your account along with a temporary password set by the system to your account. You will also get a copy of this document by email.
 - e) Log into PARAM Ganga and you will be prompted to change the password. Once you change the temporary password provided by the system to your own password, you are ready to use PARAM Ganga.

Info: *

PARAM Ganga, (IIT Roorkee)
Indian Institute of Technology, Roorkee
Email: gangasupport@iitr.ac.in

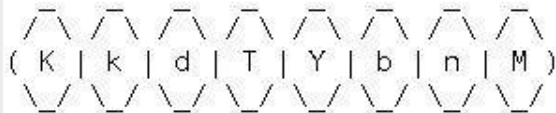
First login

Whenever the newly created user on PARAM Ganga tries to log in with the User Id and password (temporary, system generated) provided over the email through PARAM Ganga support, he/she will next be prompted to create a “new password” of their choice which will change the temporary, system generated password. This will enable you to keep your

account secure. It is recommended that you have a strong password that contains a combination of alphabets (lower case/upper case), numbers, and a few special characters that you can easily remember.

Given next is a screenshot that describes the scenario for “first login”

Observe the picture below and answer the question listed afterwards:



Type the string above: KkdTYbnM

Password:

You are required to change your password immediately (password aged)
password expired 18078 days ago

New password:

Your password will be valid for 90 days. On expiry of 90 days period, you will be prompted to change your password, on attempting to log in. You are required to provide a new password.

Forgot Password?

There is nothing to panic!! Please raise a ticket regarding this issue and the system administrators will resolve your problem. Please refer to the section “Getting Help – PARAM Ganga Support”, described elsewhere in this manual. Follow the GUI based, user-friendly ticketing system. Please follow the steps given below:

1. Open the PARAM Ganga support site i.e., the ticketing tool by following the link <https://paramganga.iitr.ac.in/support>
2. Login with your registered email id, complete name, contact number.
3. There you can raise a ticket to get the password reset.
4. The system admin person will revert with an email for verification.
5. Once acknowledged, the password is reset for the user and an email is sent back for intimating the same.
6. Then the user can login with the temporary password and can set a new password of his/her choice.

System Access

Accessing the cluster

The cluster can be accessed through 10 general login nodes, which allows users to login.

- You may access login node through ssh.
- The login node is primary gateway to the rest of the cluster, which has a job scheduler (called Slurm). You may submit jobs to the queue and they will run when the required resources are available.
- Please do not run programs directly on login node. Login node is use to submit jobs, transfer data and to compile source code. (If your compilation takes more than a few minutes, you should submit the compilation job into the queue to be run on the cluster.)
- By default, two directories are available (i.e., /home and /scratch). These directories are available on login node as well as the other nodes on the cluster. /scratch is for temporary data storage, generally used to store data required for running jobs.

Remote Access

Using SSH in Windows

To access PARAM Ganga, you need to “ssh” the login server. PuTTY is the most popular open source “ssh” client application for Windows, you can download it from (<http://www.putty.org/>). Once installed, find the PuTTY application shortcut in your Start Menu, desktop. On clicking the PuTTY icon, the PuTTY configuration dialog should appear. Locate the “Host Name or IP Address” input field in the PuTTY configuration screen. Enter the user’s name along with IP address or Hostname with which you wish to connect.

(e.g. [username]@ paramganga.iitr.ac.in -p 4422) for outside access

Enter your password when prompted, and press Enter.

Using SSH in Mac or Linux

Both Mac and Linux systems provide a built-in SSH client, so there is no need to install any additional package. Open the terminal, connect to SSH server by typing the following command:

```
ssh [username]@[hostname]
```

For example, to connect to the PARAM Ganga Login Node, with the username

```
user1: ssh user1@paramganga.iitr.ac.in -p 4422
```

You will be prompted for a password, and then will be connected to the server.

Password

How to change the user password?

Use the **passwd** command to change the password for the user from login node.

```
[nikhleshs@login1 ~]$ passwd
Changing password for user nikhleshs.
(current) LDAP Password:
New password:
Retype new password:
```

Transferring files between local machine and HPC cluster

Users need to have the data and application related to their project/research work on PARAM Ganga.

To store the data special directories have been made available to the users with the name “home” the path to this directory is “/home”. Whereas these directories are common to all the users, a user will get his own directory with their username in /home/ directories where they can store their data.

```
/home/<username>/: This directory is generally used by the user to
install applications.
```

However, there is a limit to the storage provided to the users, the limits have been defined according to quota over these directories, all users will be allotted the same quota by default. When a user wishes to transfer data from their local system (laptop/desktop) to the HPC system, they can use various methods and tools.

A user using ‘Windows’ operating system will get methods and tools that are native to Microsoft Windows and tools that could be installed on your Microsoft windows machine. Linux operating system users do not require any tool. They can just use the “scp” command on their terminal, as mentioned below.

Users are advised to keep a copy of their data with themselves, once the project/research work is completed by transferring the data in from PARAM Ganga to their local system (laptop/desktop). The command shown below can be used for effecting file transfers (in all the tools):

```
scp -P 4422 -r <path to the local data directory> <your
username>@paramganga.iitr.ac.in:<path to directory on HPC where to
save the data>
```

Example:

Same Command could be used to transfer data from the HPC system to your local system (laptop/desktop).

```
scp -r /dir/dir/file username@<cluster IP/Name>:/home/username
```

Example:

```
scp -r <path to directory on HPC> <your username>@<IP of local
system>:<path to the local data directory>
```

```
scp -r /home/testuser testuser@<local system IP/Name>:/dir/dir/file
```

Note: The Local system (laptop/desktop) should be connected to the network with which it can access the HPC system.

To reiterate,

Copying Directory/File from local machine to PARAM Ganga:

To copy a local directory from your Linux system (say Wrf-2.0) to your home directory in your PARAM Ganga HPC account, the procedure is:

1. From the terminal go to the parent directory using cd command.

```
user1@mylaptop:~$ cd ~/MyData/
```

2. Under parent directory type ls <& press Enter key>, & notice Wrf-2.0 is there.

```
user1@mylaptop: ~$ ls Files TempFiles-0.5 Wrf-2.0
```

3. Begin copy by typing:

```
user1@mylaptop:~$ scp -P 4422 -r Wrf-2.0 (username)@paramganga.iitr.ac.in  
< you will be prompted for password; enter your password >
```

4. Now login to your account as:

```
user1@mylaptop:~$ ssh (your username)@ paramganga.iitr.ac.in -p 4422  
< you will be prompted for password; enter password >  
[user1@login:~]$
```

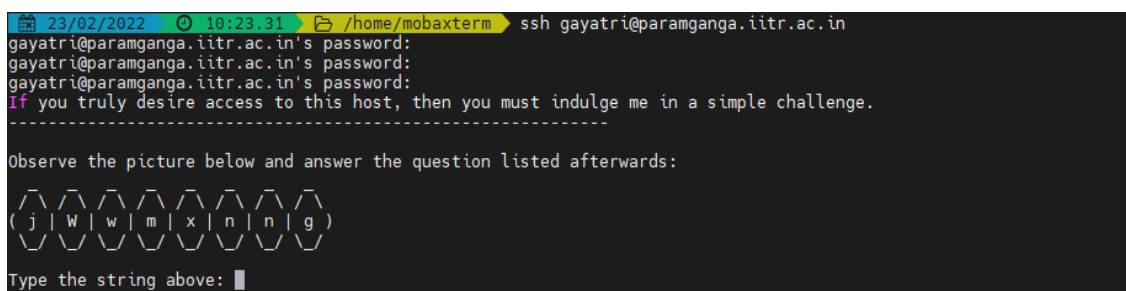
5. Type **ls** command, you should see Wrf-2.0 directory.
6. While copying from PARAM Ganga to your local machine, follow the same steps by interchanging source and destination in the scp command. Refer to the generic copying described earlier.

Tools

MobaXterm (Windows installable application):

It is a third party freely available tool which can be used to access the HPC system and transfer file to PARAM Ganga system through your local systems (laptop/desktop).

Link to download this tool: <https://mobaxterm.mobatek.net/download-home-edition.html>



```
23/02/2022 10:23:31 /home/mobaxterm ssh gayatri@paramganga.iitr.ac.in  
gayatri@paramganga.iitr.ac.in's password:  
gayatri@paramganga.iitr.ac.in's password:  
gayatri@paramganga.iitr.ac.in's password:  
If you truly desire access to this host, then you must indulge me in a simple challenge.  
-----  
Observe the picture below and answer the question listed afterwards:  
( j | W | w | m | x | n | n | g )  
Type the string above: █
```

Figure 3 - A snapshot of command using MobaXterm

WinSCP (Windows installable application)

This popular tool is freely available and is used very often to transfer data from Windows machine to Linux machine. This tool is GUI based which makes it very user-friendly.

Link for this tool is: <https://winscp.net/eng/download.php>

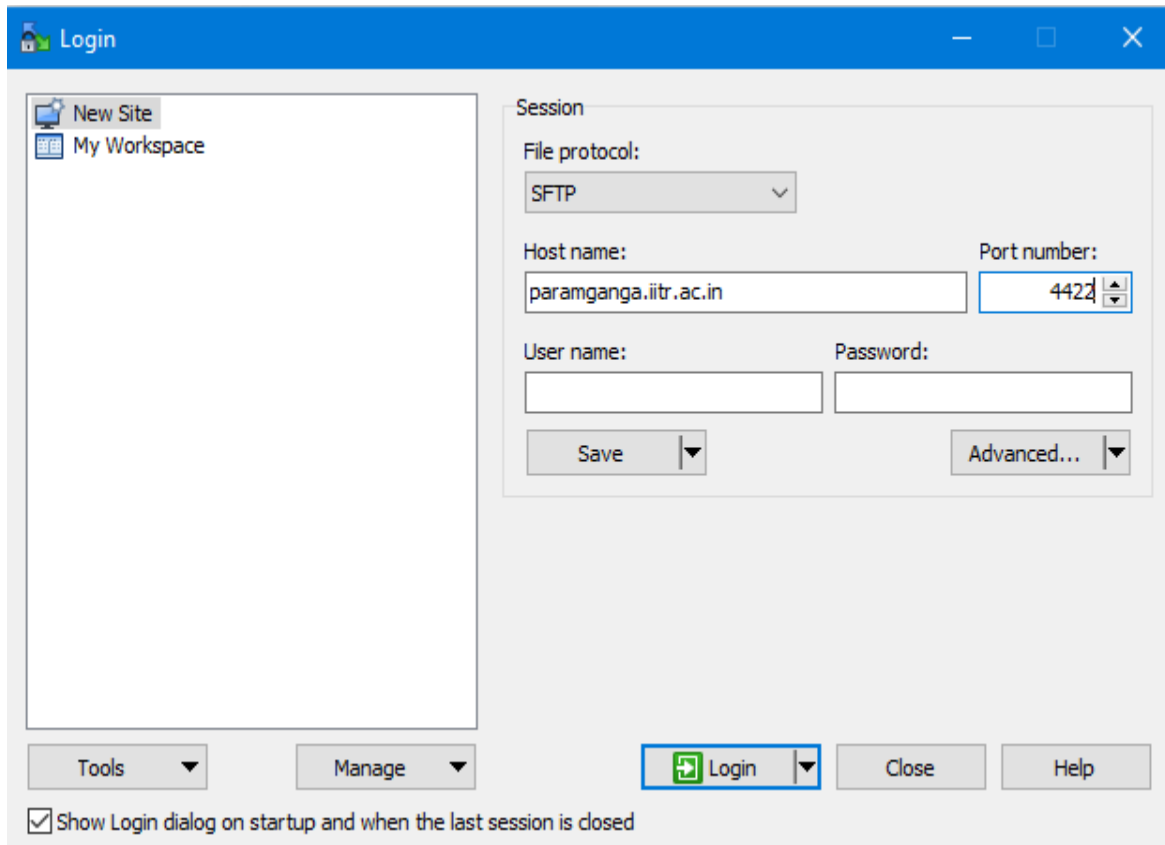


Figure 4 - A snapshot of "scp" tool to transfer file to and from remote computer.

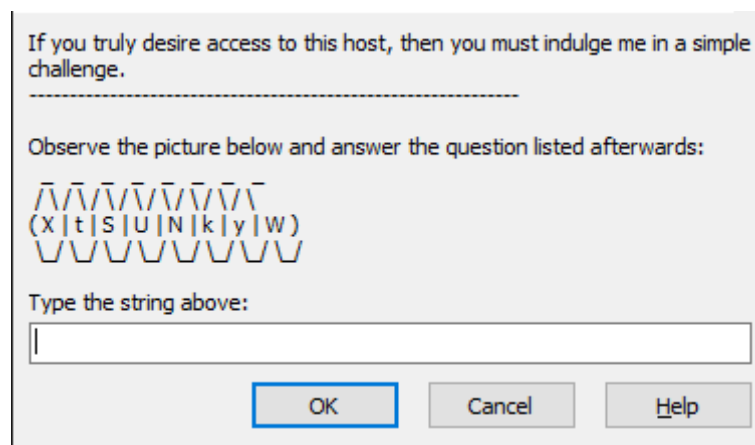


Figure 5 – Enter Captcha/String

Note: Port Used for SFTP connection is 4422 and not 22. Please change it to 4422

Running Interactive Jobs

In general, the jobs can be run in an interactive manner or in batch mode. You can run an interactive job as follows:

The following command asks for a single core on one hour with default amount of memory.

```
$ srun --nodes=1 --ntasks-per-node=1 --time=01:00:00 --pty bash -i
```

The command prompt will appear as soon as the job starts. This is how it looks once the interactive job starts:

```
srun: job xxxxx queued and waiting for resources srun: job xxxxx has been  
allocated resources
```

Where xxxxx is the job id.

Exit the bash shell to end the job. If you exceed the time or memory limits the job will also abort.

Please note that PARAM Ganga is NOT meant for executing interactive jobs. However, for the purpose of quickly ascertaining successful run of a job before submitting a large job in batch (with large iteration counts), this can be used. This can even be used for running small jobs. The point to be kept in mind is that, since others too would be using this node, it is prudent not to inconvenience them by running large jobs.

It is a good idea to specify the CPU account name as well (if you face any problems)

```
$ srun --account=<NAME_OF_MY_ACCOUNT> --nodes=1 --ntasks-per-node=1 --  
time=01:00:00 -- pty bash -i
```

Managing Jobs through its Lifecycle

PARAM Ganga extensively uses modules. The purpose of module is to provide the production environment for a given application, outside of the application itself. This also specifies which version of the application is available for a given session. All applications and libraries are made available through module files. A user has to load the appropriate module from the available modules. User can add a particular module in their `~/.bashrc` also if they don’t want to load particular module file for each time after they login.

```
module avail                                # This command lists all the available modules

module load compiler/intel/2018.4 # This will load the intel compilers into your
environment

module unload  compiler/intel/2018.4 # This will remove all environment setting related
to intel-2018 compiler loaded previously
```

A simple Slurm job script

```
#!/bin/sh
#SBATCH -N 1                                // specifies number of nodes
#SBATCH --ntasks-per-node=48 // specifies cores per node
#SBATCH --time=06:50:20 // specifies maximum duration of run
#SBATCH --job-name=lammps // specifies job name
#SBATCH --error=job.%J.err_node_48 // specifies error file name
#SBATCH --output=job.%J.out_node_48 //specifies output file name
#SBATCH --partition=standard // specifies queue name

cd $SLURM_SUBMIT_DIR // To run job in the directory from where it is
submitted
export I_MPI_FABRICS=shm:dapl //For Intel MPI versions 2019
onwards this value must be shm:ofi
mpiexec.hydra -n $SLURM_NTASKS lammps.exe
```

walltime

Walltime parameter defines as to how long your job will run. The maximum runtime of a job allowed as per QoS policy. If more than 3 days are required, a special request needs to be

sent to HPC coordinator and it will be dealt with on a case-to-case basis. The command line to specify walltime is given below.

```
srun -t walltime <days-hours:mins:seconds>
```

and also, as part of the submit scripts described in the manual. If a job does not get completed within the walltime specified in the script, it will get terminated.

The biggest advantage of specifying appropriate walltime is that the efficiency of scheduling improves resulting in improved throughput in all jobs including yours. You are encouraging to arrive at the appropriate walltime for your job by executing your jobs few times.

NOTE: You are requested to explicitly specify the walltime in your command lines and scripts.

List Partition

sinfo displays information about nodes and partitions(queues).

\$ sinfo

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
standard*	up	14-00:00:0	5	drain*	cn[001-002,113,158],hm009
standard*	up	14-00:00:0	1	down*	cn083
standard*	up	14-00:00:0	1	mix	hm010
standard*	up	14-00:00:0	216	resv	cn[003-082,084-112,114-157,159-192],gpu[001-011],hm[001-008,011-020]
gpu	up	14-00:00:0	11	resv	gpu[001-011]
cpu	up	14-00:00:0	4	drain*	cn[001-002,113,158]
cpu	up	14-00:00:0	1	down*	cn083
cpu	up	14-00:00:0	187	resv	cn[003-082,084-112,114-157,159-192]
hm	up	14-00:00:0	1	drain*	hm009
hm	up	14-00:00:0	1	mix	hm010
hm	up	14-00:00:0	18	resv	hm[001-008,011-020]

Figure 6 - Output of sinfo command

Submit the job

We can consider three cases of submitting a job

1. Submitting a simple standalone job

This is a simple submit script which is to be submitted

```
$ sbatch slurm-job.sh
Submitted batch job 106
```

2. Submit a job that's dependent on a prerequisite job being completed

Consider a requirement of pre-processing a job before proceeding to actual processing. Pre-processing is generally done on a single core. In this scenario, the actual processing script is dependent on the outcome of pre-processing script.

Here's a simple job script. Note that the Slurm -J option is used to give the job a name.

```
#!/usr/bin/env bash
#SBATCH -p standard
#SBATCH -J simple
sleep 60
Submit the job: $ sbatch simple.sh
Submitted batch job 149
```

Now we'll submit another job that's dependent on the previous job. There are many ways to specify the dependency conditions, but the "singleton" method is the simplest. The Slurm -d singleton argument tells Slurm not to dispatch this job until all previous jobs with the same name have completed.

```
$ sbatch -d singleton simple.sh //may be used for first pre-processing
on a core and then submitting
Submitted batch job 150
$ squeue
  JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
   150 standard  simple user1 PD 0:00 1 (Dependency)
   149 standard  simple user1  R 0:17 1 cn001
```

Once the prerequisite job finishes the dependent job is dispatched.

```
$ squeue
  JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
   150 standard  simple user1  R 0:31 1 cn001
```

3. Submit a job with a reservation allocated

Slurm has the ability to reserve resources for jobs being executed by select users and/or select bank accounts. A resource reservation identifies the resources in that reservation and a time period during which the reservation is available. The resources which can be reserved include cores, nodes.

Use the command given below to check the reservation name allocated to your user account

```
$ scontrol show reservation
```

If your ‘user account’ is associated with any reservation the above command will show you the same. For e.g., the reservation name given is user_11. Use the command given below to make use of this reservation

```
$ sbatch --reservation=user_11 simple.sh
```

4. Submitting multiple jobs with minor or no changes (array jobs)

A **SLURM job array** is a collection of jobs that differs from each other by only a single index parameter. Job arrays can be used to submit and manage a large number of jobs with similar settings.

```
# Submit a job array with index values between 0 and 31
$ sbatch --array=0-31 -N1 tmp

# Submit a job array with index values of 1, 3, 5 and 7
$ sbatch --array=1,3,5,7 -N1 tmp

# Submit a job array with index values between 1 and 7
# with a step size of 2 (i.e. 1, 3, 5 and 7)
$ sbatch --array=1-7:2 -N1 tmp
```

Figure 7 – Snapshot depicting the usage of “Job Array”

N1 is specifying number of nodes you want use for your job. example: N1 -one node, N4 - four nodes. Instead of tmp here you can use below example script.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --ntasks-per-node=48
#SBATCH --error=job.%A_%a.err
#SBATCH --output=job.%A_%a.out
#SBATCH --time=01:00:00
#SBATCH --partition=standard

module load compiler/intel/2018.2.199
cd /home/guest/Rajneesh/Rajneesh
export OMP_NUM_THREADS=${SLURM_ARRAY_TASK_ID}
/home/guest/Rajneesh/Rajneesh/md_omp
```

List jobs

Monitoring jobs on SLURM can be done using the command **squeue**. Squeue is used to view job and job step information for jobs managed by SLURM.

```
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
  106 standard      slurm-job user1  R   0:04      1 cn001
```

Get job details

scontrol can be used to report more detailed information about nodes, partitions, jobs, job steps, and configuration.

scontrol show node - shows detailed information about compute nodes.

```
[root@login04 ~]# scontrol show node cn001
NodeName=cn001 Arch=x86_64 CoresPerSocket=24
CPUAlloc=0 CPUTot=48 CPULoad=0.01
AvailableFeatures=cpu,centos7
ActiveFeatures=cpu,centos7
Gres=(null)
NodeAddr=cn001 NodeHostName=cn001 Version=20.11.8
OS=Linux 3.10.0-1160.el7.x86_64 #1 SMP Mon Oct 19 16:18:59 UTC 2020
RealMemory=1 AllocMem=0 FreeMem=182850 Sockets=2 Boards=1
State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=standard,cpu
BootTime=2021-12-15T12:42:46 SlurmdStartTime=2021-12-29T15:17:40
CfgTRES=cpu=48,mem=1M,billing=48
AllocTRES=
CapWatts=n/a
CurrentWatts=0 AveWatts=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
Comment=(null)
```

Figure 8 – scontrol show node displays compute node information

scontrol show partition - shows detailed information about a specific partition

```
[root@login04 ~]# scontrol show partition hm
PartitionName=hm
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=N/A
DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
Nodes=hm[001-039]
PriorityJobFactor=75 PriorityTier=75 RootOnly=NO ReqResv=NO OverSubscribe=NO
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=1872 TotalNodes=39 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

Figure 9 – scontrol show partition displays specific partition details

scontrol show job - shows detailed information about a specific job or all jobs if no job id is given.

```
[root@ananta0 ~]# scontrol show job 4253
JobId=4253 JobName=copy.sh
  UserId=atos01(40004) GroupId=atos01(40004) MCS_label=N/A
  Priority=19212 Nice=0 Account=atos QOS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:16:44 TimeLimit=02:00:00 TimeMin=N/A
  SubmitTime=2022-03-23T19:31:40 EligibleTime=2022-03-23T19:31:40
  AccrueTime=2022-03-23T19:31:40
  StartTime=2022-03-23T19:31:40 EndTime=2022-03-23T21:31:40 Deadline=N/A
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-03-23T19:31:40
  Partition=standard AllocNode:Sid=login03:91479
  ReqNodeList=gpu001 ExcNodeList=(null)
  NodeList=gpu001
  BatchHost=gpu001
  NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=1,node=1,billing=1
  Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
  MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
  Features=(null) DelayBoot=00:00:00
  OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=/mnt/scratch/atos01/test/copy.sh
  WorkDir=/mnt/scratch/atos01/test
  StdErr=/mnt/scratch/atos01/test/slurm-4253.out
  StdIn=/dev/null
  StdOut=/mnt/scratch/atos01/test/slurm-4253.out
  Power=
  NtasksPerTRES:0
```

Figure 10 – scontrol show job displays specific job information

scontrol update job - changes attributes of submitted job; like time limit, priority (root only)

```
$ scontrol show job 106
JobId=106 Name=slurm-job.sh
  UserId=user1(1001) GroupId=user1(1001)
  Priority=4294901717 Account=(null) QOS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
  RunTime=00:00:07 TimeLimit=14-00:00:0 TimeMin=N/A
  SubmitTime=2021-01-26T12:55:02 EligibleTime=2021-01-26T12:55:02
  StartTime=2021-01-26T12:55:02 EndTime=Unknown
  PreemptTime=None SuspendTime=None SecsPreSuspend=0
  Partition=standard AllocNode:Sid=atom-head1:3526
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=cn001
  BatchHost=cn001
  NumNodes=1 NumCPUs=2 CPUs/Task=1 ReqS:C:T=*:*:~
  MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
  Features=(null) Gres=(null) Reservation=(null)
  Shared=0 Contiguous=0 Licenses=(null) Network=(null)
  Command=/home/user1/slurm/local/slurm-job.sh
```

```
WorkDir=/home/user1/slurm/local
```

scontrol update job= 106 TimeLimit=04-00:00:0

Suspend a job (root only):

```
# scontrol suspend 135
# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 135 standard simple.sh user01 S 0:10 1 cn001
```

Resume a job (root only):

```
# scontrol resume 135
# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 135 standard simple.sh user01 R 0:13 1 cn001
```

Kill a job. Users can kill their own jobs; root can kill any job.

```
$ scancel 135
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
```

Hold a job:

```
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 139 standard simple user01 PD 0:00 1 (Dependency)
 138 standard simple user01 R 0:16 1 cn001
$ scontrol hold 139
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 139 standard simple user01 PD 0:00 1 (JobHeldUser)
 138 standard simple user01 R 0:32 1 cn001
```

Release a job:

```
$ scontrol release 139
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 139 standard simple user01 PD 0:00 1 (Dependency)
 138 standard simple user01 R 0:46 1 cn001
```


Addressing Basic Security Concerns

Your account on PARAM Ganga is ‘private to you’. You are responsible for any actions emanating from your account. It is suggested that you should never share the password to anyone including your friends and system administrators!!

Please note that, by default, a new account created on PARAM Ganga is readable by everyone on the system. The following simple commands will make your account adequately safe.

chmod 700 /home/\$user	! will ensure that only yourself can read, write and ! execute files in your home directory
chmod 750 /home/\$user	! will enable yourself and the members of your ! group to read and execute files in your home ! directory
chmod 755 /home/\$user	! will enable yourself, your group members and ! everyone else to read and execute files in your ! directory
chmod 777 /home/\$user	! will enable EVERY ONE on the system to read, ! write and execute files in your home directory. ! This is a sort of ‘free for all’ situation. This ! should be used very judiciously

More about Batch Jobs (SLURM)

SLURM (Simple Linux Utility for Resource Management) is a workload manager that provides a framework for job queues, allocation of compute nodes, and the start and execution of jobs.

It is important to note:

- Compilations are done on the login node. Only the execution is scheduled via SLURM on the compute nodes
- Upon Submission of a Job script, each job gets a unique Job Id. This can be obtained from the ‘**squeue**’ command.
- The Job Id is also appended to the output and error filenames.
- All examples of modules are for illustrations only, please refer to the cluster for actual module name.

Parameters used in SLURM job script

The job flags are used with SBATCH command. The syntax for the SLURM directive in a script is "#SBATCH <flag>". Some of the flags are used with the srun and salloc commands.

Resource	Flag Syntax	Description
partition	--partition=partition name	Partition is a queue for jobs.
time	--time=01:00:00	Time limit for the job.
nodes	--nodes=2	Number of compute nodes for the job.
cpus/cores	--ntasks-per-node=8	Corresponds to number of cores on the compute node.
resource feature account	--gres=gpu:2	Request use of GPUs on compute nodes
	--account=group-slurm-account	Users may belong to groups or accounts.
job name	--job-name="lammeps"	Name of job.
output file	--output=lammeps.out	Name of file for stdout.
	-w, --nodelist	Request a specific list of hosts.
	--mail-type=	Notify user by email when certain event types occur. Valid

Resource	Flag Syntax	Description
		type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit), and TIME_LIMIT_50 (reached 50 percent of time limit). Multiple type values may be specified in a comma separated list
email address	--mail-user username@cdac.in User to receive email notification of state changes as defined by --mail-type	User's email address
access	--exclusive	Exclusive access to compute nodes. The job allocation cannot share nodes with other running jobs

Script for a Sequential Job

```
#!/bin/bash
#SBATCH -N 1 // number of nodes
#SBATCH --ntasks-per-node=1 // number of cores per node
#SBATCH --error=job.%J.err // name of output file
#SBATCH --output=job.%J.out // name of error file
#SBATCH --time=01:00:00 // time required to execute the program
#SBATCH --partition=standard // specifies queue name (standard is the
default partition if you do not specify any partition job will be submitted
using default partition). For other partitions you can specify hm or gpu

// To load the module //

module load compiler/intel/2018.2.199

cd <Path of the executable>

/home/cdac/a.out (Name of the executable).
```

Script for a Parallel OpenMP Job

```
#!/bin/bash
#SBATCH -N 1 // Number of nodes
#SBATCH --ntasks-per-node=48 // Number of cores per node
#SBATCH --error=job.%J.err // Name of output file
#SBATCH --output=job.%J.out // Name of error file
```

```
#SBATCH --time=01:00:00 // Time take to execute the program #SBATCH --
partition=standard // specifies queue name(standard is the default
partition if you does not specify any partition job will be submitted using
default partition) other partitions
You can specify hm and gpu

// To load the module //

module load compiler/intel/2018.4

cd <path of the executable>
or
cd $SLURM_SUBMIT_DIR //To run job in the directory from where it is
submitted

export OMP_NUM_THREADS=48 (Depending upon your requirement you can change
number of threads. If total number of threads per node is more than 48,
multiple threads will share core(s) and performance may degrade)

/home/cdac/a.out (Name of the executable)
```

Script for Parallel Job – MPI (Message Passing Interface)

```
#!/bin/sh

#SBATCH -N 16 // Number of nodes
#SBATCH --ntasks-per-node=48 // Number of cores per node
#SBATCH --time=06:50:20 // Time required to execute the program
#SBATCH --job-name=lammps // Name of application
#SBATCH --error=job.%J.err_16_node_48 // Name of the output file
#SBATCH --output=job.%J.out_16_node_48 // Name of the error file
#SBATCH --partition=standard // Partition or queue name

// To load the module //
module load compiler/intel/2018.4

// Below are Intel MPI specific settings //
export I_MPI_FALLBACK=disable

export I_MPI_FABRICS=shm:dapl
export I_MPI_DEBUG=9 // Level of MPI verbosity //

cd $SLURM_SUBMIT_DIR
or
cd /home/manjuv/LAMMPS_2018COMPILER/lammps-22Aug18/bench

// Command to run the lammps in Parallel //

time mpiexec.hydra -n $SLURM_NTASKS -genv OMP_NUM_THREADS 1
/home/manjuv/LAMMPS_2018COMPILER/lammps-22Aug18/src/lmp_intel_cpu_intelmpi
-in in.lj
```

Script for Hybrid Parallel Job – (MPI + OpenMP)

```
#!/bin/sh

#SBATCH -N 16 // Number of nodes
#SBATCH --ntasks-per-node=48 // Number of cores for node
#SBATCH --time=06:50:20 // Time required to execute the program
#SBATCH --job-name=lammps // Name of application
#SBATCH --error=job.%J.err_16_node_48 // Name of the output file
#SBATCH --output=job.%J.out_16_node_48 // Name of the error file
#SBATCH --partition=standard // Partition or queue name

cd $SLURM_SUBMIT_DIR

// To load the module //

module load compiler/intel/2018.2.199
```

```
// Below are Intel MPI specific settings //
export I_MPI_FALLBACK=disable

export I_MPI_FABRICS=shm:dapl
```

```
export I_MPI_DEBUG=9 // Level of MPI verbosity //
export OMP_NUM_THREADS=24 //Possibly then total no. of MPI ranks will be =
(total no. of cores, in this case 16 nodes x 48 cores/node) divided by (no.
of threads per MPI rank i.e. 24)

// Command to run the lammps in Parallel //

time mpiexec.hydra -n 32 lammps.exe -in in.lj
```

I am familiar with PBS/ TORQUE. How do I migrate to SLURM?

Environment Variables	PBS/Torque	SLURM
Job Id	\$PBS_JOBID	\$SLURM_JOBID
Submit Directory	\$PBS_JOBID	\$SLURM_SUBMIT_DIR
Node List	\$PBS_NODEFILE	\$SLURM_JOB_NODELIST
Job Specification	PBS/Torque	SLURM
Script directive	#PBS	#SBATCH
Job Name	-N [name]	--job-name=[name] OR -J [name]
Node Count	-1 nodes=[count]	--nodes=[min[-max]] OR -N

		[min[-max]]
CPU count	-1 ppn=[count]	---ntasks-per-node=[count]
CPUs Per Task		--cpus-per-task=[count]
Memory Size	-1 mem-[MB]	--mem=[MB] OR – mem_per_cpu=[MB]
Wall Clock Limit	-1 walltime=[hh:mm:ss]	--time=[min] OR – mem_per_cpu=[MB]
Node Properties	-1 nodes=4.ppn=8:[property]	--constraint=[list]
Standard Output File	-o [file_name]	--output=[file_name] OR -o [file_name]
Standard Error File	-e [file_name]	--error=[file_name] OR -e {file_name}
Combine stdout/stderr	-j oe (both to stdout)	(This is default if you do not specify –error)
Job Arrays	-t [array_spec]	--array=[array_spec] OR -a [array_spec]
Delay Job Start	-a [time]	--begin=[time]

Preparing Your Own Executable

The compilations are done on the login node, whereas the execution happens on the compute nodes via the scheduler (SLURM).

Note: The Compilation and execution must be done with same libraries and matching version to avoid unexpected results.

Steps:

1. Load required modules on the login node.
2. Do the compilation.
3. Open the job submission script and specify the same modules to be loaded as used while compilation.
4. Submit the script.

The directory contains a few sample programs and their sample job submission scripts. The compilation and execution instructions are described in the beginning of the respective files.

The user can copy the directory to his/her home directory and further try compiling and executing these sample codes. The command for copying is as follows:

```
cp -r /home/apps/Docs/samples/ ~/.
```

1. mm.c - Serial Version of Matrix-Matrix Multiplication of two NxN matrices
2. mm_omp.c - Basic OpenMP Version of Matrix-Matrix Multiplication of two NxN matrices
3. mm_mpi.c - Basic MPI Version of Matrix-Matrix Multiplication of two NxN matrices
4. mm_acc.c - OpenAcc Version of Matrix-Matrix Multiplication of two NxN matrices
5. mm_blas.cu - CUDA Matrix Multiplication program using the cuBlas library.
6. mm_mkl.c - MKL Matrix Multiplication program.
7. laplace_acc.c - OpenACC version of the basic stencil problem.

It is recommended to use the intel compilers since they are better optimized for the hardware.

Compilers

Compilers	Description	Versions Available
gcc/gfortran	GNU Compiler (C/C++/Fortran)	4.8.5, 5.5.0, 7.3.0, 8.3.0, 9.3.0
icc/icpc/ifort	Intel Compilers (C/C++/Fortran)	16.x, 17.x, 18.x, 19.x
mpicc/mpicxx/mpif90	Intel-MPI with GNU compilers (C/C++/Fortran)	16.x, 17.x, 18.x, 19.x
mpiicc/mpiicpc/mpiifort	Intel-MPI with Intel compilers (C/C++/Fortran)	16.x, 17.x, 18.x, 19.x
nvcc	CUDA C Compiler	7.5, 8.0, 9.0, 9.2, 10.0, 10.1, 10.2
pgcc/pgc++/pgfortran	PGI Compiler (C/C++/Fortran)	19.4, 19.10

Optimization Flags

Optimization flags are meant for uniprocessor optimization, wherein, the compiler tries to optimize the program, on the basis of the level of optimization. The optimization flags may also change the precision of output produced from the executable. The optimization flags can be explored more on the respective compiler pages. A few examples are given below.

```
Intel: -O3 -xHost
GNU: -O3
PGI: -fast
```

Given next is a brief description of compilation and execution of the various types of programs. However, for certain bigger applications, loading of additional dependency libraries might be required.

C Program:

```
Setting up of environment: module load compiler/intel/2018.2.199
compiler/gcc/7.3.0
compilation: icc -O3 -xHost <<prog_name.c>>
Execution: ./a.out
```

C + OpenMP Program:

```
Setting up of environment: module load compiler/intel/2018.2.199
compiler/gcc/7.3.0
Compilation: icc -O3 -xHost -qopenmp <<prog_name.c>>
Execution: ./a.out
```

C + MPI Program:

```
Setting up of environment: module load compiler/intel/2018.2.199
compiler/gcc/7.3.0
Compilation: mpiicc -O3 -xHost <<prog_name.c>>
Execution: mpirun -n <<num_procs>> ./a.out
```

C + MKL Program:

```
Setting up of environment:
module load compiler/intel/2018.2.199 compiler/gcc/7.3.0
Compilation: icc -O3 -xHost -mkl <<prog_name.c>>
Execution: ./a.out
```

CUDA Program:

```
Setting up of environment:
module load compiler/cuda/10.1 compiler/gcc/7.3.0
```

Example (1)

```
Compilation: nvcc -arch=sm_70 <<prog_name.cu>>
Execution: ./a.out
```

Note: The optimization switch `-arch=sm_70` is intended for Volta V100 GPUs and is valid for CUDA 9 and later. Similarly, older versions of CUDA have compatibility with lower versions of GCC only. Accordingly, appropriate modules of GCC must be loaded.

Example (2)

```
Compilation: nvcc -arch=sm_70 /home/apps/Docs/samples/mm_blas.cu -
lcublas
Execution: ./a.out
```

CUDA + OpenMP Program:

```
Setting up of environment:
module load compiler/cuda/10.1 compiler/gcc/7.3.0
```

Example (1)

```
Compilation: nvcc -arch=sm_70 -Xcompiler="-fopenmp" -lgomp
/home/apps/Docs/samples/mm_blas_omp.cu -lcublas
Execution: ./a.out
```

Example (2)

```
Compilation: g++ -fopenmp /home/apps/Docs/samples/mm_blas_omp.c -
I/opt/ohpc/pub/apps/cuda/cuda-10.1/include -
L/opt/ohpc/pub/apps/cuda/cuda-10.1/lib64 -lcublas
Execution: ./a.out
```

OpenACC Program:

```
Setting up of environment:
module load compiler/pgi/19.10 compiler/cuda/10.1
```

```
Compilation for GPU: pgcc -acc -fast -Minfo=all -ta=tesla:cc70,managed
/home/apps/Docs/samples/laplace_acc.c
Execution: ./a.out
```

```
Compilation for CPU: pgcc -acc -fast -Minfo=all -ta=multicore -
tp=skylake /home/apps/Docs/samples/laplace_acc.c
Execution: ./a.out
```

Job Submission on Scheduler (SLURM)

A sample job submission scripts for each of the sample programs is given. Upon completion/termination of the execution, two files (output and error) are generated.

A few sample commands for SLURM are as follows:

<pre>sinfo</pre>	Lists out the status of resources in the system
<pre>squeue</pre>	Lists out the Job information in the system
<pre>sbatch <<job_script>></pre>	Submitting a job to the scheduler
<pre>scancel <<job_name>></pre>	Delete a job

Spack

Introduction

Spack automates the download-build-install process for software - including dependencies - and provides convenient management of versions and build configurations. It is designed to support multiple versions and configurations of software on a wide variety of platforms and environments. It is designed for large supercomputing centers, where many users and application teams share common installations of software on clusters with exotic architectures, using libraries that do not have a standard ABI. Spack is non-destructive: installing a new version does not break existing installations, so many configurations can coexist on the same system.

Getting Started

On your login node command prompt execute below commands:

\$ module load spack - To load SPACK module and setting up environment for SPACK.

```
[cdacadmin01@login02 ~]$ module load spack/0.16.4
For better command line support, copy and paste the following,
which will source the spack setup script:
. /home/apps/spack/share/spack/setup-env.sh
If using spack to install to system area, make sure to set umask 0002
so that group write access is available to the software Linux group.
[cdacadmin01@login02 ~]$ █
```

Kindly see the above screenshot and source below line including initial dot.

```
$ . /home/apps/spack/share/spack/setup-env.sh
```

To Use Pre-Installed Applications from Spack

spack find

The `spack find` command is used to query installed packages on PARAM Ganga. Note that some packages appear identical with the default output. The `-l` flag shows the hash of each package, and the `-f` flag shows any non-empty compiler flags of those packages.

```
[cdacadmin01@login02 ~]$ spack find
==> 3012 installed packages
-- linux-centos7-cascadelake / aocc@3.2.0 -----
autoconf@2.69      kbproto@1.0.7      libxdmcp@1.1.2     readline@8.1
automake@1.16.5    libbsd@0.11.3      libxml2@2.9.12     sqlite@3.37.1
berkeley-db@18.1.40 libevent@2.1.12    libxml2@2.9.12     tar@1.34
bzip2@1.0.8        libfabric@1.14.0   m4@1.4.19          texinfo@6.5
cmake@3.21.4       libffi@3.3         nasm@2.15.05       triangle@1.6
diffutils@3.8      libiconv@1.16      ncurses@6.2        util-linux-uuid@2.36.2
expat@2.4.3        libiconv@1.16      ncurses@6.2        util-macros@1.19.3
fftw@3.3.10        libmd@1.0.3        numactl@2.0.14     xcb-proto@1.14.1
findutils@4.8.0    libpciaccess@0.16 openblas@0.3.19     xextproto@7.3.0
freetype@2.11.0    libpng@1.6.37      openmpi@4.1.2      xproto@7.0.31
gdbm@1.19          libpthread-stubs@0.4 openssl@7.4p1       xtrans@1.3.5
gettext@0.21       libsigsegv@2.13    openssl@1.1.1l     xz@5.2.2
gmake@4.3          libtool@2.4.6      openssl@1.1.1m     xz@5.2.5
hwloc@2.6.0        libx11@1.7.0       perl@5.34.0        zlib@1.2.11
hwloc@2.7.0        libxau@1.0.8       pkgconf@1.8.0      zlib@1.2.11
inputproto@2.3.2   libxcb@1.14        pkgconf@1.8.0
-- linux-centos7-cascadelake / gcc@11.2.0 -----
adios2@2.7.1       py-dill@0.3.4
alsa-lib@1.2.3.2   py-dill@0.3.4
amdfftw@3.0        py-dill@0.3.4
aocc@3.2.0         py-dill@0.3.4
```

spack load application name

The easiest way is to use `spack load <application name@version>`

```
[cdacadmin01@login02 ~]$ spack load gromacs@2021.3
==> Error: gromacs@2021.3 matches multiple packages.
Matching packages:
  wgffx3h gromacs@2021.3%gcc@11.2.0 arch=linux-centos7-cascadelake
  7dwaixh gromacs@2021.3%intel@2021.4.0 arch=linux-centos7-cascadelake
  pbuwbkj gromacs@2021.3%oneapi@2021.4.0 arch=linux-centos7-cascadelake
  3wq4ngs gromacs@2021.3%gcc@11.2.0 arch=linux-centos7-cascadelake
  csyksoy gromacs@2021.3%gcc@11.2.0 arch=linux-centos7-cascadelake
  qwppbij gromacs@2021.3%gcc@11.2.0 arch=linux-centos7-cascadelake
Use a more specific spec.
[cdacadmin01@login02 ~]$ spack load gromacs@2021.3/ wgffx3h
[cdacadmin01@login02 ~]$ which gmx_mpi
/home/apps/spack/opt/spack/linux-centos7-cascadelake/gcc-11.2.0/gromacs-2021.3-wgffx3h/imflbg76up435qk
ufqtvt7ljbr/bin/gmx_mpi
[cdacadmin01@login02 ~]$
```

To know the Pre-Loaded Application/Compilers

```
$ spack find --loaded
```

```

==> 15 loaded packages
-- linux-centos7-cascadelake / gcc@11.2.0 -----
fftw@3.3.10      libedit@3.1-20210216  libpciaccess@0.16  numactl@2.0.14
openssl@1.1.11
gromacs@2021.3  libevent@2.1.12      libxml2@2.9.12    openmpi@4.1.1
xz@5.2.5
hwloc@2.6.0    libiconv@1.16        ncurses@6.2       openssh@8.7p1
zlib@1.2.11

```

To install new application

First check the available compilers in Spack with below command:

spack compilers

Spack manages a list of available compilers on the system, detected automatically from the user’s PATH variable. The spack compilers command is an alias for the command spack compiler list.

```

[cdacadmin01@login02 ~]$ spack compilers
==> Available compilers
-- gcc centos7-x86_64 -----
gcc@11.2.0  gcc@8.3.0  gcc@4.8.5

-- intel centos7-x86_64 -----
intel@2021.4.0

-- oneapi centos7-x86_64 -----
oneapi@2021.4.0
[cdacadmin01@login02 ~]$ █

```

To check the compilers available in the system

```

$ spack compiler list
==> Available compilers
-- gcc centos7-x86_64 -----
gcc@11.2.0  gcc@8.3.0  gcc@4.8.5

-- intel centos7-x86_64 -----
intel@2021.4.0

-- oneapi centos7-x86_64 -----
oneapi@2021.4.0

```

Check if application is available in Spack repo with command-

spack list

The spack list command shows available packages.

The spack list command can also take a query string. Spack automatically adds wildcards to both ends of the string, or you can add your own wildcards.

```
[cdacadmin01@login02 ~]$ spack list | more
3dtk
3proxy
abduco
abi-compliance-checker
abi-dumper
abinit
abseil-cpp
abyss
accfft
acct
accumulo
ace
ack
acl
acpica-tools
acpid
activeharmony
activemq
acts
addrwatch
adept-utils
adf
adiak
adios
--More--
```

Before installing application check its spec with command

spack install

Below is an example of installation of package using spack:

```
spack install gromacs@2020.5 +cuda~mpi+blas %intel ^intel-mkl
```

Above command will install gromacs version 2020.5 with blas and cuda support and without MPI support. For blas there are multiple providers like OpenBLAS, Intel MKL, amdblis, and essl, ^intel-mkl will tell spack to use intel-mkl for blas routines.

Operators in Spack

- % to select compiler out of available compilers
- ^ to use variant of package
- @ to define the version number of packages to be installed.
- + to enable variant for package
- ~ to disable variant for package

Uninstalling Packages

Earlier we installed many configurations each of zlib. Now we will go through and uninstall some of those packages that we didn’t really need.

```
$ spack uninstall zlib %gcc@6.5.0
(type: y)
```

Using Environments

Spack has an environment feature in which you can group installed software. You can install software with different versions and dependencies in each environment and can change software to use at once by changing environments. You can create a Spack environment by **spack env create** command. You can create multiple environments by specifying different environment names here.

```
spack env create myenv
```

To activate the created environment, type `spack env activate`. Adding `-p` option will display the current activated environment on your console. Then, install software you need to the activated environment.

```
spack env activate -p myenv
myenv] [username@es1 ~]$ spack install xxxxx
```

You can deactivate the environment by `spack env deactivate`. To switch to another environment, type `spack env activate` to activate it.

```
[myenv] [username@es1 ~]$ spack env deactivate
[username@es1 ~]$
```

Use `spack env list` to display the list of created Spack environments.

```
[username@es1 ~]$ spack env list
==> 2 environments
    myenv
    another_env
```

Packaging (For Application developers)

Spack packages are installation scripts, which are essentially recipes for building the software.

They define properties and behaviour of the build, such as:

- where to find and how to retrieve the software.
- its dependencies.
- options for building the software from source; and
- build commands.

Once we’ve specified a package’s recipe, users of our recipe can ask Spack to build the software with different features on any of the supported systems. Please refer [Packaging Guide — Spack 0.17.0 documentation](#) for detailed understanding of the Spack packaging.

Example Creating Own Package:

In below spec file we have used **Linewidth an IISc developed code**. Please see the bold lines for comments related to preceding lines in the spec file of spack package named `liscLinewidth`:

```
# Copyright 2013-2021 Lawrence Livermore National Security, LLC and other
# Spack Project Developers. See the top-level COPYRIGHT file for details.
#
# SPDX-License-Identifier: (Apache-2.0 OR MIT)
import os
import platform
import sys
import llnl.util.tty as tty
from spack import *
class IiscLinewidth(MakefilePackage):
    """
        Linewidth developed by IISC Bangalore.
    """
    homepage = ""
    #Url for homepage
    url      = "file://{0}/linewidth.tar.gz".format(os.getcwd())
    #Url for source code
    manual_download = True
    #If source code is not available in public domain
    version('1',
sha256='7215f6765e5f5eddfde5f0c67a5bbdef5960607f3e199a609ef5619278ec8a66',
        preferred=True)
```



```

#You can add different versions for you package.
variant('mpi', default=True, description='Install with MPI support')
variant('openmp', default=True, description='Install with OpenMP
support')
#Variant gives flexibility to users for changing parameter before
compilation.
depends_on('gmake', type='build')
depends_on('mpi', when='+mpi')
depends_on('hdf5+fortran+hl+mpi')
depends_on('intel-mkl')
depends_on('py-h5py')
depends_on('py-matplotlib', type=('build', 'run'))
#Depend clause used to specify dependencies for your code.

@property
def build_targets(self):
    targets = [
        # '--directory=SRC',
        '--file=Makefile',
        'LIBS={0} {1} '.format(self.spec['intel-mkl'].libs.ld_flags,
                               self.spec['hdf5'].libs.ld_flags),
        'HDFINCFLAGS={0}'.format(self.spec['hdf5'].prefix.include),
        'HDF5_HOME={0}'.format(self.spec['hdf5'].prefix),
        'FC={0}'.format(self.spec['mpi'].mpifc)
    ]
    return targets
def install(self, spec, prefix):
    mkdirp(prefix.bin)
    install('linewidth', prefix.bin)
####
#This code uses Makefile for building application. We can define some
properties
# to make changes in Makefile, changing parameter in Makefile at compile
time.

```

Sample steps taken for creating linewidth application recipe for Spack

1. Source code
Source code of Linewidth was not available through public repo like GitHub, so needed to import OS package.
os.getcwd() - expects the source tar present in current working directory.
cha256 - to check for sha256 checksum we added same in version clause and for place holder we have given version as 1.
manual download = True refers to spack will not try to download source code for the package.
name - make sure that name of tar file is same as used inside package recipe
2. Variant- User can control behavior of application being built through this clause.
Ex- To enable MPI support we have defined it to be true by default.
3. depends_on() - This clause defines all dependencies required to build the given application.
Ex- In linewidth example we have used Intel-MKL and HDF5.

4. @property - With this decorator we can define some properties for build system like edit, build, install.
5. property build_targets - Defines logic of building source for native platform.
6. property install - Defines install procedure to be used after building source code.
Ex- In our example we define prefix path

Sample SLURM script for OpenMP applications/programs. to use Spack

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH -p cpu ## gpu/standard
#SBATCH --exclusive
#SBATCH -t 1:00:00

echo "SLURM_JOBID"=${SLURM_JOBID}
echo "SLURM_JOB_NODELIST"=${SLURM_JOB_NODELIST}
echo "SLURM_NNODES"=${SLURM_NNODES}
echo "SLURM_NTASKS"=${SLURM_NTASKS}
ulimit -s unlimited
ulimit -c unlimited
export OMP_NUM_THREADS=4 ### Maximum number of threads= Number of physical
core

#To load necessary application/compiler through spack
module load spack
export SPACK_ROOT=/home/apps/spack
. $SPACK_ROOT/share/spack/setup-env.sh
spack load intel-mpi@2019.10.317 /6icwzn3
spack load intel-mkl@2020.4.304
spack load intel-oneapi-compilers@2021.4.0
spack load gcc@11.2.0

(time <executable_path>)
```

Sample SLURM script for MPI applications/programs to use Spack

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH -p cpu ## gpu/standard
#SBATCH --exclusive
#SBATCH -t 1:00:00

echo "SLURM_JOBID"=${SLURM_JOBID}
echo "SLURM_JOB_NODELIST"=${SLURM_JOB_NODELIST}
echo "SLURM_NNODES"=${SLURM_NNODES}
echo "SLURM_NTASKS"=${SLURM_NTASKS}
ulimit -s unlimited
ulimit -c unlimited
```

```
#To load necessary application/compiler through spack
module load spack
export SPACK_ROOT=/home/apps/spack
. $SPACK_ROOT/share/spack/setup-env.sh
spack load intel-mpi@2019.10.317 /6icwzn3
spack load intel-mkl@2020.4.304
spack load intel-oneapi-compilers@2021.4.0
spack load gcc@11.2.0
(time mpirun -np $SLURM_NTASKS <executable_path>)
```

Job Scheduling on PARAM Ganga

Scheduler

PARAM Ganga has Slurm-20.11.08 (open source) as a workload manager for HPC facility. Slurm is a widely used batch scheduler in top500 HPC list. PARAM Ganga consists of three types of compute nodes: i.e., CPU only (192 GB) nodes, High memory (768 GB) nodes and Nvidia GPU (192 GB) enabled.

Following partitions/queues have been defined for different requirements -

1. **standard**: CPU, High memory and GPU Jobs
2. **gpu**: GPU and CPU jobs
3. **hm**: High memory intensive jobs

All users can submit to the standard partition. The standard partition contains CPU, high memory and GPU nodes. GPU partition contains only gpu nodes. If user wants to submit a job only on gpu nodes, he/she can use gpu partition. If user wants to submit a job only on high memory, he/she can use hm partition.

Note: User has to specify `#SBATCH --gres=gpu:1/2` in their job script if user wants to use 1 or 2 GPU cards on GPU nodes

sinfo

This Slurm command is used to view available **partition** and **node** information on the cluster.

```

standard* up 4-00:00:00 0 down* cn[010],gpu[003-005,007-008]
standard* up 4-00:00:00 2 unk* cn[011,027]
standard* up 4-00:00:00 1 mix gpu001
standard* up 4-00:00:00 150 idle cn[001-010,012-015,017-026,028-110],gpu[002,006,009-010],hm[001-039]
gpu up 4-00:00:00 5 down* gpu[003-005,007-008]
gpu up 4-00:00:00 1 mix gpu001
gpu up 4-00:00:00 4 idle gpu[002,006,009-010]
hm up 4-00:00:00 39 idle hm[001-039]
cpu up 4-00:00:00 1 down* cn016
cpu up 4-00:00:00 2 unk* cn[011,027]
cpu up 4-00:00:00 107 idle cn[001-010,012-015,017-026,028-110]
debug up 1-00:00:00 1 down* cn016
debug up 1-00:00:00 2 unk* cn[011,027]
debug up 1-00:00:00 146 idle cn[001-010,012-015,017-026,028-110],hm[001-039]
small up 3-00:00:00 1 down* cn016
small up 3-00:00:00 2 unk* cn[011,027]
small up 3-00:00:00 146 idle cn[001-010,012-015,017-026,028-110],hm[001-039]
medium up 1-00:00:00 1 down* cn016
medium up 1-00:00:00 2 unk* cn[011,027]
medium up 1-00:00:00 146 idle cn[001-010,012-015,017-026,028-110],hm[001-039]
large up 1-00:00:00 1 down* cn016
large up 1-00:00:00 2 unk* cn[011,027]
large up 1-00:00:00 146 idle cn[001-010,012-015,017-026,028-110],hm[001-039]
highmemory up 1-00:00:00 39 idle hm[001-039]
gpumultinode up 1-00:00:00 5 down* gpu[003-005,007-008]
gpumultinode up 1-00:00:00 1 mix gpu001
gpumultinode up 1-00:00:00 4 idle gpu[002,006,009-010]
gpusinglenode up 1-00:00:00 5 down* gpu[003-005,007-008]
gpusinglenode up 1-00:00:00 1 mix gpu001
gpusinglenode up 1-00:00:00 4 idle gpu[002,006,009-010]
hip-small up 1-00:00:00 1 down* cn016
hip-small up 1-00:00:00 2 unk* cn[011,027]
hip-small up 1-00:00:00 146 idle cn[001-010,012-015,017-026,028-110],hm[001-039]
hip-large up 1-00:00:00 1 down* cn016
hip-large up 1-00:00:00 2 unk* cn[011,027]
hip-large up 1-00:00:00 146 idle cn[001-010,012-015,017-026,028-110],hm[001-039]
hip-gpu up 1-00:00:00 5 down* gpu[003-005,007-008]
hip-gpu up 1-00:00:00 1 mix gpu001
hip-gpu up 1-00:00:00 4 idle gpu[002,006,009-010]

```

Figure 11 – sinfo Command

PARAM Ganga SLURM Partitions and QoS

NAME	Priority	Min Core/gpu	Max core/gpu	Max Walltime (HH:MM:SS)	Max Submit jobs per user	Max Running Job per user	Overall Running jobs
DEBUG	4000	128	256	01:00:00	3	2	15
SMALL	3000	128	512	03-00:00:00	3	2	25
MEDIUM	3700	526	4048	24:00:00	2	1	2

LARGE	4000	4056	10146	24:00:00	1	1	1
GPU	2000	1 GPU	40 GPU	24:00:00	5	3	10
HIGHMEMORY	3000	24	3720	24:00:00	4	3	10

For Submitting the job

Partition Name	Srun command (x=number of nodes)	Sbatch command (x=number of nodes)	Script
DEBUG	<code>srun -Nx -p debug --pty /bin/bash</code>	<code>sbatch -Nx -p debug samplescript.sh</code>	<code>#SBATCH --partition=debug</code>
SMALL	<code>srun -Nx -p small --pty /bin/bash</code>	<code>sbatch -Nx -p small samplescript.sh</code>	<code>#SBATCH --partition=small</code>
MEDIUM	<code>srun -Nx -p medium --pty /bin/bash</code>	<code>sbatch -Nx -p medium samplescript.sh</code>	<code>#SBATCH --partition=medium</code>
LARGE	<code>srun -Nx -p large --pty /bin/bash</code>	<code>sbatch -Nx -p large samplescript.sh</code>	<code>#SBATCH --partition=large</code>
GPU	<code>srun -Nx --gres=gpu:2 -p gpusinglenode --pty /bin/bash</code>	<code>sbatch -Nx --gres=gpu:2 -p gpusinglenode samplescript.sh</code>	<code>#SBATCH --partition=gpusinglenode</code> <code>#SBATCH --gres=gpu:number of gpu</code>

NOTE: For gpu partition use `--gres=gpu:number of gpu`

debug

The debug limit is intended for testing, not for production throughput. Users are limited to minimum 128 core and maximum 256 cores per job. The maximum walltime per job will be 1 hour and only 3 jobs per user (2 running and 1 jobs in queue) are allowed for this partition. Users may also wish to compile their codes on this partition.

small

This partition has a maximum run time of 72 hours (3 days). This will be the default partition in case if no partition name is specified in the job script. It is designed to run small jobs with limit of minimum 128 cores and maximum 512 cores. At given instance maximum 25 jobs can be in running state and only 3 jobs per user are allowed (2 running and 1 in queue).

medium

The medium partition has a maximum run time of 24 hours (1 day). The minimum cores required for this partition are 526 and the maximum cores can be up to 4048 cores per job. Only 2 jobs can be in running state with this partition. Each user can submit maximum 2 jobs in this partition (2 running).

large

The large partition has maximum core limit with combination of cpu and hm cores. The users have walltime limit of 24 hours (1 day) with 4056 minimum and maximum 10146 cores. Only one job is allowed per user.

gpu

This partition is specifically for accessing gpu nodes with 2 Nvidia V100 GPUs per node. Users need to add the flag **#SBATCH --gres=gpu** to gain access to the nodes. The maximum time limit is 24 hrs (1 day) and maximum only 2 gpus can be accessed with this partition.

walltime

Walltime parameter defines as to how long your job will run. The maximum runtime of a job allowed as per the QoS Policy. If more than 4 days are required, a special request needs to be sent to HPC coordinator and it will be dealt with on a case-to-case basis. The command line to specify walltime is given below.

```
srun -t walltime <days-hours:mins:seconds>
```

and also, as part of the submit scripts described in the manual. If a job does not get completed within the walltime specified in the script, it will get terminated.

The biggest advantage of specifying appropriate walltime is that the efficiency of scheduling improves resulting in improved throughput in all jobs including yours. You are encouraging to arrive at the appropriate walltime for your job by executing your jobs few times.

Note: Default wall time is 2 hours, you have to specify wall time if you want the job to run more than 2 hours.

NOTE: You are requested to explicitly specify the walltime in your command lines and scripts.

Per user

- Every user will have quota of 48G of soft limit and 50G of hard limit with grace period of 7 days in HOME file system (/home) and 190G of soft limit and 200G of hard limit with grace period of 14 days in SCRATCH file system
- Users are recommended to copy their execution environment and input files to scratch file system (/scratch/<username>) during job running and copy output data back to HOME area
- File retention policy has been implemented on Lustre storage for the "/scratch" file system. As per the policy, any files that have not been accessed for the last 3 months will be deleted permanently
- Three QoS (Quality of services) are created according to different job sizes and wall time. Resource limits for users are defined as per below QoS policy

Scheduling Type

PARAM Ganga has been configured with Slurm's backfill scheduling policy. It is good for ensuring higher system utilization; it will start lower priority jobs if doing so does not delay the expected start time of any higher priority jobs. Since the expected start time of pending jobs depends upon the expected completion time of running jobs, reasonably accurate time limits are important for backfill scheduling to work well.

Job Priority

The job's priority at any given time will be a weighted sum of all the factors that have been enabled in the slurm.conf file. Job priority can be expressed as:

```
Job_priority =
  (PriorityWeightAge) * (age_factor) +
  (PriorityWeightFairshare) * (fair-share_factor) +
  (PriorityWeightJobSize) * (job_size_factor) +
  (PriorityWeightPartition) * (partition_factor) +
  (PriorityWeightQOS) * (QOS_factor) +
  SUM(TRES_weight_cpu * TRES_factor_cpu,
      TRES_weight_<type> * TRES_factor_<type>,
      ...)
```


All of the factors in this formula are floating point numbers that range from 0.0 to 1.0. The weights are unsigned, 32-bit integers. The job's priority is an integer that ranges between 0 and 4294967295. The larger the number, the higher the job will be positioned in the queue, and the sooner the job will be scheduled. A job's priority, and hence its order in the queue, can vary over time. For example, the longer a job sits in the queue, the higher its priority will grow when the age weight is non-zero.

Age Factor: The age factor represents the length of time a job has been sitting in the queue and eligible to run. Current value for Age factor is 10000.

Job Size Factor: The job size factor correlates to the number of nodes or CPUs the job has requested. Current value for Job Size factor is 1000.

Partition Factor: Each node partition can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request to run in this partition. Current value for partition factor is 15000.

Quality of Service (QoS) Factor: Each QoS can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request this QoS. Current value for QoS factor is 100000.

Fair-share Factor: The fair-share component to a job's priority influences the order in which a user's queued jobs are scheduled to run based on the portion of the computing resources they have been allocated and the resources their jobs have already consumed. Current value for fair-share factor is 100000.

```
[root@cpu-login1 ~]# sshare
```

Account	User	RawShares	NormShares	RawUsage	EffectvUsage	FairShare
root			1.000000	1338680274	1.000000	0.500000
root	root	1	0.009901	9038	0.000007	0.999527
c-dac		50	0.495050	1288517	0.000963	0.998652
itg		50	0.495050	1337382717	0.999030	0.246893
bio	parent		0.495050	0	0.999030	0.246893
biotechnology	parent		0.495050	0	0.999030	0.246893
chemical	parent		0.495050	32376927	0.999030	0.246893
chemistry	parent		0.495050	40427024	0.999030	0.246893
civil	parent		0.495050	464979	0.999030	0.246893
cse	parent		0.495050	147	0.999030	0.246893
eee	parent		0.495050	0	0.999030	0.246893
environment	parent		0.495050	7711556	0.999030	0.246893
mathematics	parent		0.495050	0	0.999030	0.246893
mechanical	parent		0.495050	19699157	0.999030	0.246893
nanotechnology	parent		0.495050	0	0.999030	0.246893
physics	parent		0.495050	1236702925	0.999030	0.246893

```
[root@cpu-login1 ~]# █
```

Figure 12 - Listing the shares of association to a cluster

ACCOUNTING

Accounting system tracks and manages HPC resource usage. As jobs are completed or resources are utilized, accounts are charged and resource usage is recorded. Accounting policy is like a bank/Credit System, where each department can be allocated with some pre-defined budget on a quarterly basis for CPU usage. As and when the resources are utilized, the amount will be deducted. The allocation will be reset at end of every quarter.

sacct

This command can report resource usage for running or terminated jobs including individual tasks, which can be useful to detect load imbalance between the tasks.

sstat

This command can be used to status only currently running jobs.

sreport

This command can be used to generate reports based upon all jobs executed in a particular time interval.

Standard priority queue

CPU xp/minute/core ! Useful for charging calculations GPU/minute/accelerator ! Useful for charging calculations

High Priority queue

CPU xx/minute/core ! Useful for charging calculations GPU/minute/accelerator ! Useful for charging calculations

Storage Policy

FileSystem	Size	Quota	Access	Retention Period
/home	~576 TiB	500GB	RW	Unlimited
/scratch	~1728 TiB	1TB	RW	60 days

Debugging Your Codes

Introduction

A **debugger** or **debugging tool** is a computer program that is used to test and debug other programs (the "target" program).

When the program "traps" or reaches a preset condition, the debugger typically shows the location in the original code if it is a source-level debugger or symbolic debugger, commonly now seen in **integrated development environments**.

Debuggers also offer more sophisticated functions such as running a program step by step (single-stepping or program animation), stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint, and tracking the values of variables.

Some debuggers have the ability to modify program state while it is running. It may also be possible to continue execution at a different location in the program to bypass a crash or logical error.

Basics How Tos

Compilation

Compilation with a separate flag '-g' is required since the program needs to be linked with debugging symbols.

```
gcc -g <program_name.c>  
e.x. gcc -g random_generator.c
```

Running with gdb

`gdb` is a command line utility available with almost all Linux systems' compiler collection packages.

```
gdb <executable.out>  
e.x. gdb a.out
```

Basic gdb Commands (to be executed in gdb command line window)

Start:

Starts the program execution and stops at the first line of the main procedure. Command line arguments may be provided if any.

Run:

Starts the program execution but does not stop. It stops only when any error or program trap occurs. Command line arguments may be provided if any.

Help:

Prints the list of command available. Specifying 'help' followed by a command (e.x. 'help run') displays more information about that command.

File <filename>:

Loads a binary program that is compiled with '-g' flag for debugging.

List [line_no]:

Displays the source code (nearby 10 lines) of the program in execution where the execution stopped. If 'line_no' is specified, it display the source code (10 lines) at the specified line.

Info:

Displays more information about the set of utilities and saved information by the debugger. For example; 'info breakpoints' will list all the breakpoints, similarly 'info watchpoints' will list all the watch points set by the user while debugging their programs.

Print <expression>:

Prints the values of variables / expression at the current running instance of the program.

Step N:

Steps the program one (or 'N') instructions ahead or till the program stops for any reason. Steps through each and every instruction even if it is function call (only function or instruction compiled with debugging flags).

next:

This command also steps through the instructions of the program. Unlike 'step' command, if the current source code line calls a subroutine, this command does not enter the subroutine, but instead steps over the call, if effect treating it as a single source line.

Continue:

This command continues the stopped program till the next breakpoint has occurred or till the end of the program. It is used to continue from a paused/debug point state.

Break [sourcefile:] <line_no> [if condition]:

Stops the program at the specified line number and provides a breakpoint for the user. Specific source code file and breakpoint based on a condition can also be set for specific cases. You can also view the list of breakpoints set, by using the 'info breakpoints' command.

watch <expression>:

A watchpoint means break the program or stop the execution of the program when the value of the expression provided is changed. Using watch command specific variables can be watched for value changes. You can also view the list of watchpoints by using the 'info watchpoints' command.

Delete <breakpoint number>

Delete command deletes a breakpoint or a watchpoint that has been set by a user while debugging the program.

Backtrace:

Prints the backtrace of all stack frames of the program. Provides the call stack and more other information about the running program.

These are some of the most powerful utilities that can be used to debug your programs using gdb. gdb is not limited to these commands and contains a rich set of features that can allow you to debug multi-threaded programs as well. Also, all the commands, along with the ones listed above have 'n' number of different variants for more in-depth control. Same can be utilized using the help page of gdb.

Using gdb (example – inspecting the code)

For this case study, we have a small program that generates a long unique random number for each run.

Let's look at the code we have.

```

#include <stdio.h>           //printf
#include <stdlib.h>         //malloc, srand, rand
#include <unistd.h>         //getpid

#define N 100
#define N_LEN 100

//Generate a short random number
short rand_fract(void) {
    short sum = 1;
    for (short i = 0; i < (rand() % N); ++i)
        for (short j = 0; i < N; ++j) {
            int value = (i * j) / (i + j);
            sum += (value != 0) ? value : sum;
        }
    return sum;
}

//Returns the factorial of a number
long long factorial(unsigned int x) {
    if (x == 1 || x == 0)
        return 1LL;
    else
        return (x * factorial(x - 1));
}

```

Figure 13 – Snapshot of debugging process

Things to note:

- 1) We have a few libraries included for the functions that are used in the program.
- 2) We have two '#define' statements:
 - a. 'N' for the number of times the 'rand_fract' function will spend in calculating the random number.
 - b. 'N_LEN' for the length of the final random number string generated. Currently it is set to '100' which means that the long random number will be of length 100.
- 3) Then, we have a function by name 'rand_fract' that iterates over two loops and using the values of iterators ('i' and 'j'), it calculates a small random number. Since, 'rand()' function is used for the outer loop, its number of iterations cannot be clearly defined which gives the function a random nature.

- 4) The next function is as simple as its name is. It just takes an unsigned integer and returns its factorial.

PART 2:

```
int main (int argc, char *argv[]) {
    short f1 = 0;

    //Create a random seed based on process id.
    srand((unsigned int) getpid());

    //Generate a random number salt.
    f1 = rand_fract() % 10;

    //Get the factorial of the number
    long long random_fact = factorial(f1);

    //Normalize the factorial to number modulo N_LEN + 1
    int normalized_fact = random_fact % (N_LEN + 1);

    int *array = NULL;

    //Create an array of size obtained from normalized factorial modulo N_LEN + 1
    array = (int *) malloc (sizeof (int) * normalized_fact);
    if (array == NULL) { printf("Not enough memory\n"); return -1; }

    //Populate the array with integers ni reverse order
    //Double the number five times if it is even
    for (int i = 0; i < normalized_fact; ++i) {
        array[i] = (normalized_fact - i);
    }

    //Print the serial number
    for (int i = normalized_fact - 1; i >= 0; --i)
        printf("%0d", (array[i] + rand()) % 10);
    for (int i = (N_LEN - normalized_fact); i > 0; --i)
        printf("%0d", (rand() % 10));
    printf("\n");

    //Free allocated memory
    free(array);

    return 0;
}
```

Figure 14 – Snapshot of debugging process

Things to note:

- 1) This is the main function of the program.
- 2) The flow of the main function is as follows:
 - a. The program first sets a random seed using the process-id of the program.

- b. It calls ‘rand_fract’ function and the resultant random number is operated by a modulo 10 operations. Finally, the result is stored in the variable ‘f1’.
- c. Next the factorial of the obtained ‘f1’ is calculated and stored in ‘random_fract’.
- d. This result is again passed through a modulo ‘N_LEN + 1’ and stored in ‘normalized_fact’.
- e. Then a dynamic array is constructed and partially filled with integer values in descending order from the ‘normalized_fact’ value.
- f. Finally, the partial array is printed by mixing the value of the array with rand() function values followed by a modulo 10 operation.
- g. The remaining partial part of final random value is generated using a basic rand () modulo 10 operations.

Using gdb (example – using the debugger)

The code that we looked upon seems correct, as well as it compiles successfully without any errors. But, when we run this code snippet, this is the result we get.

```
$ gcc random_generator.c
$ ./a.out
Floating point exception (core dumped)
$ █
```

Figure 15- Output at a debugging stage

The program ended up with a core dump without giving much information but just ‘Floating point exception’. Now let’s compile the code with debugging information and run the program simply with gdb.


```

$ gcc -g random_generator.c
$ gdb a.out
GNU gdb (GDB) Fedora 8.3.50.20190824-25.fc31
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) set style enabled off
(gdb) run
Starting program: /home/vineetm/debugger/a.out

Program received signal SIGFPE, Arithmetic exception.
0x00000000004011cc in rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) █

```

Figure 16 – Snapshot of debugging process

Here we compiled the code using '-g' and then used the 'run' command we studied earlier for running the program. You can observe that the debugger stopped at line number 13 where the 'Floating point exception (SIGFPE)' occurred. At this point we can even go and check the code at line number 13. But for now, let's check what other information we can get from the debugger. Let's check the values of the variables 'i' and 'j' at this point.

```

(gdb) run
Starting program: /home/vineetm/debugger/a.out

Program received signal SIGFPE, Arithmetic exception.
0x00000000004011cc in rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) print i
$1 = 0
(gdb) print j
$2 = 0
(gdb) █

```

Figure 17 – Output depicting “Arithmetic Exception”

The values of both ‘i’ and ‘j’ appear to be ‘0’ and thus a **divide by zero** exception is what caused our program to terminate. Let’s update the code such that the value of ‘i’ and ‘j’ will never become ‘0’. This is the modified code:

```
//Generate a short random number
short rand_fract(void) {
    short sum = 1;
    for (short i = 1; i < (rand() % N); ++i)
        for (short j = 1; i < N; ++j) {
            int value = (i * j) / (i + j);
            sum += (value != 0) ? value : sum;
        }
    return sum;
}
```

Figure 18 – Snapshot of debugging process

Thus, we just updated the loop index variables to start from ‘1’ instead of ‘0’. **Thus, using gdb, it was very simple to identify the point where the error occurred.** Let’s re-run our updated code and check what we get.

```
$ gcc random_generator.c
$ ./a.out
Floating point exception (core dumped)
$ █
```

Figure 19 – Well, we dumped core!!

WHAT!? This is unexpected. We just cured the error part of our program and still getting an FPE. Let’s go through the debugger and check where the error point is right now.

```

$ gcc -g random_generator.c
$ gdb a.out
GNU gdb (GDB) Fedora 8.3.50.20190824-25.fc31
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) set style enabled off
(gdb) run
Starting program: /home/vineetm/debugger/a.out

Program received signal SIGFPE, Arithmetic exception.
0x00000000004011cc in rand_fract () at random_generator.c:13
13         int value = (i * j) / (i + j);
(gdb) print i
$1 = 1
(gdb) print j
$2 = -1
(gdb) list
8         //Generate a short random number
9         short rand_fract(void) {
10            short sum = 1;
11            for (short i = 1; i < (rand() % N); ++i)
12                for (short j = 1; i < N; ++j) {
13                    int value = (i * j) / (i + j);
14                    sum += (value != 0) ? value : sum;
15                }
16            return sum;
17        }
(gdb) █

```

Figure 20 - Snapshot of debugging process

The debugger output shows that the error occurred on the same line as earlier. But in this case, the value of 'i' and 'j' are not '0,0' but they are '1, -1' which is causing the denominator at line 13 to be '0' and thus, causing an FPE. In addition to print commands, we have also issued the 'list' command which shows the nearby 10 lines of the code where the program stopped.

You can observe that some bugs in the programs are easier to debug but some aren't.

We will have to dig in much more to find out what is going on. Also, to be noted, we have our inner loop iterating from 1 to N (which is 100), but still the value of ‘j’ is printed out to be ‘-1’. How is this even possible!? Smart programmers would have the problem identified, but let’s stick to the basics on how to gdb. Let us use the ‘break’ command and set a breakpoint at line number 13 and observe what is going on.

```
(gdb) list 13
8      //Generate a short random number
9      short rand_fract(void) {
10         short sum = 1;
11         for (short i = 1; i < (rand() % N); ++i)
12             for (short j = 1; j < N; ++j) {
13                 int value = (i * j) / (i + j);
14                 sum += (value != 0) ? value : sum;
15             }
16         return sum;
17     }
(gdb) break 13
Breakpoint 1 at 0x4011b5: file random_generator.c, line 13.
(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint      keep y   0x00000000004011b5  in rand_fract at random_generator.c:13
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/vineetm/debugger/a.out

Breakpoint 1, rand_fract () at random_generator.c:13
13                 int value = (i * j) / (i + j);
(gdb) print i
$3 = 1
(gdb) print j
$4 = 1
(gdb) █
```

Figure 21 – Setting Breakpoint

Thus, using the command ‘break 13’ we have set the breakpoint at line number 13 which was verified using the ‘info breakpoint’ command. Then, we reran the program with the ‘run’ command. At line 13 the program stopped and using ‘print’ command we checked the values of ‘i’ and ‘j’. at this point, all seems to be well. Now, let’s proceed further. For stepping 1 instruction we can use the ‘step’ command. Let’s do that and observe the value of ‘j’.

```

(gdb) print j
$5 = 1
(gdb) step
14          sum += (value != 0) ? value : sum;
(gdb) step
12          for (short j = 1; i < N; ++j) {
(gdb) step

Breakpoint 1, rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) print j
$6 = 2
(gdb) step
14          sum += (value != 0) ? value : sum;
(gdb) step
12          for (short j = 1; i < N; ++j) {
(gdb) step

Breakpoint 1, rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) print j
$7 = 3
(gdb) █

```

Figure 22 – single stepping through to catch error!!

You can observe the usage of the 'step' command. We are going through the program line by line and checking the values of the variable 'j'.

There seems to be a lot of writing/typing of the 'step' command just to proceed with the program. Since, we have already set a breakpoint at line 13, we can use another command called as 'continue'. This command continues the program till the next breakpoint or the end of the program.

```

(gdb) continue
Continuing.

Breakpoint 1, rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) print j
$8 = 4
(gdb) continue
Continuing.

Breakpoint 1, rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) print j
$9 = 5
(gdb) continue
Continuing.

Breakpoint 1, rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) print j
$10 = 6
(gdb) █

```

Figure 23 – Debugging continued

You can see that we reduced the typing of 'step' command by 3 times to a 'continue' command just 1 time. But this is also having us write 'continue' and 'print' multiple times. Let us use some other utility in gdb known as 'data breakpoints' also known as watchpoints. But before that, let us delete the existing breakpoint using the 'delete' command.

```

(gdb) info breakpoints
Num      Type           Disp Enb Address          What
1        breakpoint     keep y  0x00000000004011b5 in rand_fract at random_generator.c:13
breakpoint already hit 6 times
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) █

```

Figure 24 – Debugging continued

Now let us see how to set a watchpoint.

```
(gdb) watch j
Hardware watchpoint 2: j
(gdb) info watchpoints
Num      Type          Disp Enb Address      What
2        hw watchpoint  keep y      j
(gdb) continue
Continuing.

Hardware watchpoint 2: j

Old value = 6
New value = 7
0x00000000004011f5 in rand_fract () at random_generator.c:12
12          for (short j = 1; i < N; ++j) {
(gdb)
Continuing.

Hardware watchpoint 2: j

Old value = 7
New value = 8
0x00000000004011f5 in rand_fract () at random_generator.c:12
12          for (short j = 1; i < N; ++j) {
(gdb)
Continuing.

Hardware watchpoint 2: j

Old value = 8
New value = 9
0x00000000004011f5 in rand_fract () at random_generator.c:12
12          for (short j = 1; i < N; ++j) {
(gdb) █
```

Figure 25 – Setting a watch point

Thus, using the command ‘watch j’ we have set a watchpoint over ‘j’. Now every time when the value of ‘j’ changes, a break will occur. You can also note the old and new values of ‘j’ printed out at each break. Another point to note is that after having one ‘continue’ command, the program had a break. Further, by just pressing the ‘Enter/Return’ button on the keyboard, the continue command was repeated. Thus, by pressing the ‘Enter/Return’

button, the last command is repeated. At this point, we have learned much about the debugger, but we are still not able to proceed fast with our error. Is there any other way to proceed? Well, yes!!

We want to observe at the point where the value of ‘j’ reaches closer to ‘N i.e., 100’. Which means that we are only concerned about what happens after ‘j’ reaches 99. Here, we land up on using what is called as conditional breakpoints. First, we will delete our watchpoint and then make use of the conditional breakpoint.

```
(gdb) info watchpoints
Num      Type           Disp Enb Address          What
2        hw watchpoint  keep y          j
          breakpoint already hit 4 times
(gdb) delete 2
(gdb) list 13
8         //Generate a short random number
9         short rand_fract(void) {
10        short sum = 1;
11        for (short i = 1; i < (rand() % N); ++i)
12        for (short j = 1; i < N; ++j) {
13        int value = (i * j) / (i + j);
14        sum += (value != 0) ? value : sum;
15        }
16        return sum;
17    }
(gdb) break random_generator.c:13 if j == 99
Note: breakpoint 3 also set at pc 0x4011b5.
Breakpoint 4 at 0x4011b5: file random_generator.c, line 13.
(gdb) continue
Continuing.

Breakpoint 3, rand_fract () at random_generator.c:13
13        int value = (i * j) / (i + j);
(gdb) print j
$12 = 99
(gdb) █
```

Figure 26 – Debugging continued

You can observe another variant of the ‘break’ command. We have explicitly stated the file and the line number along with a condition to stop. This is useful, when the source code is large and having multiple files. After setting a conditional break, we stopped at the point where the value of ‘j’ becomes ‘99’. Now, let us see what happens next. Since, this is a critical point at which we could observe the program, it is better if we step in the program using the ‘step’ command instead of relying on any break/watch points.


```

(gdb) print j
$17 = 99
(gdb) step
14         sum += (value != 0) ? value : sum;
(gdb)
12         for (short j = 1; i < N; ++j) {
(gdb)
13         int value = (i * j) / (i + j);
(gdb) print j
$18 = 100
(gdb) step
14         sum += (value != 0) ? value : sum;
(gdb)
12         for (short j = 1; i < N; ++j) {
(gdb)
13         int value = (i * j) / (i + j);
(gdb) print j
$19 = 101
(gdb) █

```

Figure 27 – Well, back to square one!!

This, is unexpected!! The value of 'j' should never be 100 or anything above it.

Thus, something is wrong with the conditional statement!!

By observation, we have figured out that the condition is itself wrong. It should have been 'j < N' instead of 'i < N'. This is a silly mistake of the programmer that led us to this much of an effort.

Also, the value of 'j' which was observed as '-1' was an outcome of the 'short' datatype overflow i.e., the value of 'j' went from 1 to 32767 (assuming short as 2 bytes) and then from -32768 to -1.

Finally, a hard programming bug was discovered. Let us correct this error and rerun the program.

```

$ gcc random_generator.c
$ ./a.out
Segmentation fault (core dumped)
$ ./a.out
1648815196934936907712847411075269363872465178968652936899126642679968327854843818024803725602089977
$ ./a.out
Segmentation fault (core dumped)
$ ./a.out
5697819555377639608368302418588269943918647330492449391532502328545856833586737093122407957112268963
$ ./a.out
6150930494475890863050318719122734582864309765193799040843958123681888230308039318234438024068348747
$ ./a.out
Segmentation fault (core dumped)
$ █

```

Figure 28 – Again Dumping Core!! Things are getting interesting or frustrating or both !!

This is strange!!

Sometimes the program is getting the correct output, but sometimes, we are getting a segmentation fault. Debugging such a program may be tricky since the occurrence of the bug is low. We will proceed with our standard debugger steps to identify the error.

```

$ gcc -g random_generator.c
$ gdb a.out
GNU gdb (GDB) Fedora 8.3.50.20190824-25.fc31
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) set style enabled off
(gdb) run
Starting program: /home/vineetm/debugger/a.out
5411371059776263605409873043180521681086975694174815924540859823191291008689026600122878853935366497
[Inferior 1 (process 61832) exited normally]
(gdb) █

```

Figure 29 – Debugging continued

We compiled the code and ran it using the debugger. But the program completed successfully. Let us rerun it till a point where the program fails.

```

(gdb) run
Starting program: /home/vineetm/debugger/a.out
5411371059776263605409873043180521681086975694174815924540859823191291008689026600122878853935366497
[Inferior 1 (process 61832) exited normally]
(gdb) run
Starting program: /home/vineetm/debugger/a.out
7919846386128432134671007571802513619267000845358948048917009272836772572766214308134147179016591178
[Inferior 1 (process 61978) exited normally]
(gdb) run
Starting program: /home/vineetm/debugger/a.out

Program received signal SIGSEGV, Segmentation fault.
0x00000000040126c in factorial (x=4294792703) at random_generator.c:24
24      return (x * factorial(x - 1));
(gdb) █

```

Figure 30 – Debugging continued

Here we observe a point where the program exited at the function ‘factorial’.

This is a point where the debugger didn’t give much information about what the value of the variable ‘x’ was. It just pointed out that the program failed at the function named ‘factorial’. That’s it!

Another reason for such kind of output would be because of the recursive nature of the function. The stack frame where the function ‘factorial’ failed could be in a long nest of recursive calls. At such points, it would be better to inspect the program at an earlier point and look for errors. Let us have a breakpoint before the ‘factorial’ function was called and view the value of the parameters that are passed to the function.

```

(gdb) list main
22         return 1LL;
23     else
24         return (x * factorial(x - 1));
25     }
26
27     int main (int argc, char *argv[]) {
28
29         short f1 = 0;
30
31         //Create a random seed based on process id.
(gdb)
32         srand((unsigned int) getpid());
33
34         //Generate a random number salt.
35         f1 = rand_fract() % 10;
36
37         //Get the factorial of the number
38         long long random_fact = factorial(f1);
39
40         //Normalize the factorial to number modulo N_LEN + 1
41         int normalized_fact = random_fact % (N_LEN + 1);
(gdb) break 36
Breakpoint 1 at 0x4012da: file random_generator.c, line 38.
(gdb) run
Starting program: /home/vineetm/debugger/a.out

Breakpoint 1, main (argc=1, argv=0x7fffffff0d8) at random_generator.c:38
38         long long random_fact = factorial(f1);
(gdb) print f1
$1 = 1
(gdb) continue
Continuing.
9962554943440906583333593426000827274699155147995250801174774876796185292736525250533642241728519329
[Inferior 1 (process 62328) exited normally]
(gdb) █

```

Figure 31 – Debugging continued (Will it ever end?)

Thus, we have set a breakpoint before the call of the function ‘factorial’ and ran the program. For the value of ‘f1 = 8’ for the ‘factorial’ function the process seems to exit normally. Let us rerun.

```

(gdb) run
Starting program: /home/vineetm/debugger/a.out

Breakpoint 1, main (argc=1, argv=0x7fffffff0d8) at random_generator.c:38
38         long long random_fact = factorial(f1);
(gdb) print f1
$1 = -8
(gdb) continue
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x00000000040126c in factorial (x=4294792699) at random_generator.c:24
24         return (x * factorial(x - 1));
(gdb) █

```

Figure 32 – We are almost there!!

Unexpectedly, we have got the value of 'f1' as '-8' and the program seems to have crashed. Let us observe the 'rand_fract' function and 'factorial' function once again. And study the behavior of the functions where we could get a negative number.

```
(gdb) list rand_fract
4
5     #define N 100
6     #define N_LEN 100
7
8     //Generate a short random number
9     short rand_fract(void) {
10        short sum = 1;
11        for (short i = 1; i < (rand() % N); ++i)
12            for (short j = 1; j < N; ++j) {
13                int value = (i * j) / (i + j);
(gdb)
14                sum += (value != 0) ? value : sum;
15            }
16        return sum;
17    }
18
19    //Returns the factorial of a number
20    long long factorial(unsigned int x) {
21        if (x == 1 || x == 0)
22            return 1LL;
23        else
(gdb) run
Starting program: /home/vineetm/debugger/a.out

Breakpoint 1, main (argc=1, argv=0x7fffffff0d8) at random_generator.c:38
38        long long random_fact = factorial(f1);
(gdb) print f1
$2 = -8
(gdb) █
```

Figure 33 – Debugging continued

Important points here to observe are:

The 'rand_fract' function is returning a datatype of 'short' while the calculation of the return value could be significantly large which may overflow the size of 'short', thus, causing a negative answer.

The function 'factorial' is expecting a value of type 'unsigned int'. Since the value passed to the function is a negative value, having an implicit conversion from a negative number to an unsigned number means that we are having a very large value passed to the factorial function.

Also, since the ‘factorial’ function is recursive, passing a very large number to it could cause multiple calls to the same function and thus, overflowing the stack provided to the user.

Now let us, step further into our program and see whether what we are discussing is the same behavior that is being observed.

```
(gdb) print f1
$4 = -8
(gdb) step
factorial (x=4294967288) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967287) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967286) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967285) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967284) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb) █
```

Figure 34 – At last a clue!!!

This is what we had expected!!

A number ‘-1’ passed to the ‘factorial’ function is being implicitly converted to a very large number ‘4294967295’.

Stepping in more reveals the recursive behavior of the 'factorial' function i.e., each call is having a sub call to the same function with one value less. Thus, what to do in these types of cases. Assume you have a large code where these functions are called from multiple locations.

Modifying the signature of any of the function means changing the code everywhere where the function is called. This is not affordable!! These are some cases, where a choice is to be made where patching the code is necessary for semantics of the program.

Let us observe a piece of code where this change can be made and then test our program for the expected results.

```
int main (int argc, char *argv[]) {  
  
    short f1 = 0;  
  
    //Create a random seed based on process id.  
    srand((unsigned int) getpid());  
  
    //Generate a random number salt.  
    f1 = rand_fract() % 10;  
  
    f1 = abs(f1);  
  
    //Get the factorial of the number  
    long long random_fact = factorial(f1);  
  
    //Normalize the factorial to number modulo N_LEN + 1  
    int normalized_fact = random_fact % (N_LEN + 1);  
  
    int *array = NULL;
```

Figure 35 - Correction applied!!

By observing the code, we find out that the expected value of 'f1' is between '0 to 9' (because of the modulo 10 operation).

Thus, without changing the signature of any function, we have inserted a patch (the highlighted) portion, that maintains the semantics of the code as well cures the problem that we had. Now let us just run and check our final program.

```

$ gcc random_generator.c
$ ./a.out
1947155904444356260827867895829013940560127574392384362061544757042318542200659899527743928595211645
$ ./a.out
0929989745546167856100961512939018573760223504833542534886091294243732854126729096261941760801537820
$ ./a.out
0244202592758390536991444038465396583053516022410228562188134665524049393105566500577005828487059653
$ ./a.out
28727182932285670545393680969066437379893671576029177909795701346393295764931536773483363035181911
$ ./a.out
9128766061538956027759598074797832715087451437704122190965898083361413690723150214543517739636518290
$ ./a.out
4700580792312412551673394453147630608790931492649027378923259025287077290331618510470262819931652479
$ ./a.out
3597977632870365479023130705918446909083470263354375991983675631252252710058384841530848408963208645
$ ./a.out
0864510419056291282368845079139095210792697191764209304803037158672651132052448868790301906812889064
$ ./a.out
4972916609538445900529958158240849030612776510222275380497441425328380877450674923651890544608240290
$ ./a.out
9528608642866177753983842182285047120984190000785095691019238964676666205506776407087180325311790389
$ █

```

Figure 36 – Resolved!!!

Thus, we are getting the correct results as expected.

Conclusions

We started with a program that we assumed to be functional but then the program ended up with bugs that were not straightforward. We then explored the power of the debugger and the various ways to identify the bugs in our program. We looked upon the easy solutions, and slowly migrated towards the type of bugs that are not easily traceable.

Finally, we identified and corrected all the bugs in our program with the help of the debugger and arrived at a bug free code.

Points to Note

- Bugs in the program cannot be necessarily a compilation error.
- One type of error can be caused by multiple bugs in the same line of code.
- Sometimes, it is not possible to change the code even when the problem is identified. The best way to cure this is to study the behavior of the code and apply patches wherever necessary.
- Using simple utilities from the ‘GNU Debugger’ can help in getting rid of problem causing bugs in large programs.

Overall Coding Modifications Done

```

random_generator.c                                     random_generator_buggy.c
//Generate a short random number                     //Generate a short random number
short rand_fract(void) {                             short rand_fract(void) {
  short sum = 1;                                     short sum = 1;
  for (short i = 1; i < (rand() % N); ++i)           for (short i = 0; i < (rand() % N); ++i)
    for (short j = 1; j < N; ++j) {                 for (short j = 0; j < N; ++j) {
      int value = (i * j) / (i + j);                int value = (i * j) / (i + j);
      sum += (value != 0) ? value : sum;             sum += (value != 0) ? value : sum;
    }
  return sum;
}
//Returns the factorial of a number                 //Returns the factorial of a number
long long factorial(unsigned int x) {               long long factorial(unsigned int x) {
  if (x == 1 || x == 0)                             if (x == 1 || x == 0)
    return 1LL;                                     return 1LL;
  else
    return (x * factorial(x - 1));
}
int main (int argc, char *argv[]) {                int main (int argc, char *argv[]) {
  short f1 = 0;
  //Create a random seed based on process id.
  srand((unsigned int) getpid());
  //Generate a random number salt.
  f1 = rand_fract() % 10;
  f1 = abs(f1);
  //Get the factorial of the number
  long long random_fact = factorial(f1);
}
Unicode (UTF-8) C Ln 68, Col 2                      Unicode (UTF-8) C Ln 11, Col 1

```

Figure 37 – What all we did to get things right!

Machine Learning / Deep Learning Application Development

Most of the popular python-based machine learning/deep learning libraries are installed on PARAM Ganga system. While developing and testing their applications, users have option to choose different environment / runtime setup like “virtual environment-based python libraries” or “conda runtime-based python libraries”.

For most of the major environment (virenv, conda) different modules are prepared. Users can check the list of the modules by using “**module avail**” command. Shown below is an example of loading conda environment in current bash shell and continue with application development.

Once logged into PARAM Ganga HPC Cluster, check which all libraries are available, loaded in current shell. To check list of modules loaded in current shell, use the command given below:

```
$ module list
```

To check all modules available on the system, but not loaded currently, use the command given below:

```
$ module avail
```

To activate conda environment on PARAM Ganga, load module “conda-python/3.7” as shown below:

```
$ module load conda-python/3.7
```

Conda environment has been installed with most of the popular python packages as shown below:

Tensorflow	Tensorflow-gpu	Mpi4py	Keras
Theano	Scipy	Scikit-Learn	Pytorch

Once “conda-python/3.7” module is loaded, end-users can use all libraries inside their python program. Many other modules based on virtual env are available on the system. Users can load those libraries using “module load” command and use them for their applications.

How to Install your own Software?

There are two approaches to install software.

1. System wide installation
2. Local installation.

System wide installation can be done by only admin. If you wish to do this, please approach system administrator. User can do local installation in their home directory. In this section we are describing the installation of HMMER application in user’s home directory.

Local installation

Step 1. Login to Ganga cluster by using your credential.

Step 2. Download the software that you want to install. For example, to download HMMER software use the command given below:

```
$ wget http://eddylab.org/software/hmmer/hmmer.tar.gz
```

Step 3. Untar the file. (If your software in zip format use unzip command)

```
$ tar zxf hmmer.tar.gz
```

Step 4. go to the software folder.

```
$ cd hmmer-3.3
```

Step 5. configure the installation path.

```
$ ./configure --prefix /your/install/path
```

Step 6. now run the 'make' command for install the software on installation path.

```
$ make
```

The newly compiled binaries are now in the src directory.

Step 7. Run a test suite that checks for errors in the software (optional)

```
$ make check
```

Step 8. run 'make install' to install the programs and man pages in your location mention in step 2

```
$ make install
```

By default, programs are installed in **/usr/local/bin** and man pages in **/usr/local/share/man/man1/**, if you do not provide installation path in step 2.

- * This is general instruction for installation, please refer the installation instruction or manual or readme file that comes with software for more details.
- # If you get any dependency error, resolve that or ask system admin to install that dependency if not installed.

Reference link: <http://hmmer.org/documentation.html>

Some Important Facts

About File Size

The global /home is served by a number of storage arrays. Each of the storage array contains a portion of the global /home. The size of a disk in the storage array is 285TB. Technically, the size of a file can be about 285 TB (which is really big). However, since the disk is shared by a large number of files, effectively the size of a single file will be far smaller. Normally, this file size is kept to be about few GBs which is sufficient for most of the users. However, if you wish to have file sizes which are larger than this, you need to create files across disks and this process is known as ‘striping’.

```
$ lfs setstripe -c 4 .
```

After this has been done all new files created in the current directory will be spread over 4 storage arrays each having 1/4th of the file. The file can be accessed as normal no special action needs to be taken. When the striping is set this way, it will be defined on a per directory basis so different directories can have different stripe setups in the same file system, new sub-directories will inherit the striping from its parent at the time of creation.

We recommend users to set the stripe count so that each chunk will be approx. 200-300GB each, for example:

File Size	Stripe count	Command
500 - 1000 GB	4	lfs setstripe -c 4 .
1000 – 2000 GB	8	lfs setstripe -c 8

Once a file is created with a stripe count, it cannot be changed. A user by themselves is also able to set stripe size and stripe count for their directories and a user can check the set stripe size and stripe count with command:

```
$ lfs getstripe <path to the direcorey>
```

To set the stripe count as

```
$ lfs setstripe -c 4 -s 10m <path to the direcory>
```

The options on the above command used have these respective functions.

- **-c** to set the stripe count; 0 means use the system default (usually 1) and -1 means stripe over all available OSTs (Lustre Object Storage Targets).
- **-s** to set the stripe size; 0 means use the system default (usually 1 MB) otherwise use k, m or g for KB, MB or GB respectively

Little-Endian and Big-Endian issues?

By and large, most of the computers follow little-endian format. This essentially means that the last byte of the binary representation of data is stored first. However, there is another way of representing data (used in some machines) where in the first byte of the binary representation of data is stored first. When binary files are to be read across these different kinds of machines, bytes need to be re-ordered. Many compilers do support this feature. Please explore this aspect, if a perfectly working code on a given machine, fails to get executed of another machine (with a different processor).

Best Practices for HPC

1. Do **NOT** run any job which is longer than a few minutes on the login nodes. Login node is for compilation of job. It is best to run the job on compute nodes. (compute nodes)
2. It is **recommended** to go through the beginner's guide in `/home/apps/Docs/samples`. This should serve as a good starting point for the new users.
3. Use the same compiler to compile different parts/modules/library-dependencies of an application. Using different compilers (e.g., `pgcc + icc`) to compile different parts of application may cause linking or execution issues.
4. Choosing appropriate compiler switches/flags/options (e.g. `-O3`) may increase the performance of application substantially (accuracy of output must be verified). Please refer to documentation of compilers (online / docs present inside compiler installation path / man pages etc.)
5. Modules/libraries used for execution should be the same as that used for compilations. This can be specified in the Job submission script.
6. Be aware of the amount of disk space utilized by your job(s). Do an estimate before submitting multiple jobs.
7. Please submit jobs preferably in `$SCRATCH`. You can back up your results/summaries in your `$HOME`
8. `$SCRATCH` is NOT backed up! Please download all your data to your Desktop/Laptop.
9. Before installing any software in your home, ensure that it is from a reliable and safe source. Ransomware is on the rise!
10. Please do not use spaces while creating the directories and files.
11. Please inform PARAM Ganga support when you notice something strange - e.g., unexpected slowdowns, files missing/corrupted etc.

Installed Applications/Libraries

Following is the list of few of the applications from various domains of science and engineering installed in the system.

HPC Applications	Bio-informatics	MUMmer, HMMER, MEME, Schrodinger, PHYLIP, mpiBLAST, ClustalW,
	Molecular Dynamics	NAMD (for CPU and GPU), LAMMPS, GROMACS
	Material Modeling, Quantum Chemistry	Quantum-Espresso, Abinit, CP2K, NWChem,
	CFD	OpenFOAM, SU2
	Weather, Ocean, Climate	WRF-ARW, WPS (WRF), ARWPost (WRF), RegCM, MOM, ROMS
Deep Learning Libraries	cuDNN, TensorFlow, Tensorflow with Intel Python, Tensorflow with GPU, Theano, Caffe, Keras, numpy, Scipy, Scikit-Learn, pytorch.	
Visualization Programs	GrADS, ParaView, VisIt, VMD	
Dependency Libraries	NetCDF, PNETCDF, Jasper, HDF5, Tcl, Boost, FFTW	

Standard Application Programs on PARAM Ganga

The purpose of this section is to expose the users to different application packages which have been installed. Users interested in exploring these packages may kindly go through the scripts, typical input files and typical output files. It is suggested that, at first, the users may submit the scripts provided and get a feel of executing the codes. Later, they may change the parameters and the script to meet their application requirements.

LAMMPS Applications

LAMMPS is an acronym for **L**arge-scale **A**tomic/ **M**olecular **M**assively **P**arallel **S**imulator. This is extensively used in the fields of Material Science, Physics, Chemistry and may others. More information about LAMMPS may please be found at <https://lammmps.sandia.gov> .

1. The LAMMPS input is **in.lj** file which contains the below parameters.

Input file = in.lj

```
# 3d Lennard-Jones melt

variable          x index 1
variable          y index 1
variable          z index 1

variable          xx equal 64*$x
variable          yy equal 64*$y
variable          zz equal 64*$z

units             lj
atom_style        atomic

lattice           fcc 0.8442
region            box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box        1 box
create_atoms      1 box
mass              1 1.0

velocity          all create 1.44 87287 loop geom

pair_style        lj/cut 2.5
pair_coeff         1 1 1.0 1.0 2.5

neighbor          0.3 bin
neigh_modify      delay 0 every 20 check no

fix               1 all nve

run               1000000
```

2. THE LAMMPS RUNNING SCRIPT

```
#!/bin/sh

#SBATCH -N 8
#SBATCH --ntasks-per-node=40
#SBATCH --time=08:50:20
```

```

#SBATCH --job-name=lammps
#SBATCH --error=job.%J.err_8_node_40
#SBATCH --output=job.%J.out_8_node_40
#SBATCH --partition=standard

module load compiler/intel/2018.2.199
module load compiler/intel-mpi/mpi-2018.2.199

module load compiler/gcc/7.3.0

source
/opt/ohpc/pub/apps/intel/2018_2/compilers_and_libraries_2018.2.199/linux/mkl/bin/mklvars.sh intel64

export I_MPI_FALLBACK=disable
export I_MPI_FABRICS=shm:ofa
#export I_MPI_FABRICS=shm:tmi
#export I_MPI_FABRICS=shm:dapl
export I_MPI_DEBUG=5

cd /home/manjunath/NEW_LAMMPS/lammps-7Aug19/bench

export OMP_NUM_THREADS=1

time mpiexec.hydra -n $SLURM_NTASKS -genv OMP_NUM_THREADS 1
/home/manjunath/NEW_LAMMPS/lammps-7Aug19/src/lmp_intel_cpu_intelmpi -in
in.lj

```

3. LAMMPS OUTPUT FILE.

```

LAMMPS (7 Aug 2019)
  using 1 OpenMP thread(s) per MPI task
Lattice spacing in x,y,z = 1.6796 1.6796 1.6796
Created orthogonal box = (0 0 0) to (107.494 107.494 107.494)
  5 by 8 by 8 MPI processor grid
Created 1048576 atoms
  create_atoms CPU = 0.00387692 secs
Neighbor list info ...
  update every 20 steps, delay 0 steps, check no
  max neighbors/atom: 2000, page size: 100000
  master list distance cutoff = 2.8
  ghost atom cutoff = 2.8
  binsize = 1.4, bins = 77 77 77
  1 neighbor lists, perpetual/occasional/extra = 1 0 0
  (1) pair lj/cut, perpetual
      attributes: half, newton on
      pair build: half/bin/atomonly/newton
      stencil: half/bin/3d/newton
      bin: standard
Setting up Verlet run ...
  Unit style      : lj
  Current step   : 0
  Time step      : 0.005
Per MPI rank memory allocation (min/avg/max) = 3.154 | 3.156 | 3.162 Mbytes
Step Temp E_pair E_mol TotEng Press

```

```

0          1.44   -6.7733681          0   -4.6133701   -5.0196704
1000000    0.65684946   -5.7123998          0   -4.7271266   0.49078272
Loop time of 2955.97 on 320 procs for 1000000 steps with 1048576 atoms

```

Performance: 146145.063 tau/day, 338.299 timesteps/s
99.4% CPU use with 320 MPI tasks x 1 OpenMP threads

MPI task timing breakdown:

Section	min time	avg time	max time	%varavg	%total
Pair	1284.2	1512.3	1866.9	494.3	51.16
Neigh	178.94	207.58	261.09	217.8	7.02
Comm	793.59	1207.7	1468.3	654.3	40.86
Output	0.00011516	0.00084956	0.0027411	0.0	0.00
Modify	19.566	22.639	29.863	67.3	0.77
Other		5.744			0.19

```

Nlocal:      3276.8 ave 3325 max 3231 min
Histogram:  4 7 21 63 67 80 50 22 5 1
Nghost:      5011.29 ave 5063 max 4956 min
Histogram:  5 9 26 45 57 76 51 34 12 5
Neighs:      122781 ave 127005 max 118605 min
Histogram:  3 5 36 59 63 52 66 24 11 1

```

```

Total # of neighbors = 39290074
Ave neighs/atom = 37.4699
Neighbor list builds = 50000
Dangerous builds not checked
Total wall time: 0:49:15

```

GROMACS APPLICATION

GROMACS

GRONingen MAchine for Chemical Simulations (GROMACS) is a [molecular dynamics](#) package mainly designed for simulations of [proteins](#), [lipids](#), and [nucleic acids](#). It was originally developed in the Biophysical Chemistry department of [University of Groningen](#), and is now maintained by contributors in universities and research centers worldwide. GROMACS is one of the fastest and most popular software packages available, and can run on [central processing units](#) (CPUs) and [graphics processing units](#) (GPUs).

Input description of Gromacs

Input file can be download from

ftp://ftp.gromacs.org/pub/benchmarks/water_GMX50_bare.tar.gz

The mdp option used is pme with 50000 steps

Submission Script:

```
#!/bin/sh
```

```

#SBATCH -N 10
#SBATCH --ntasks-per-node=48
##SBATCH --time=03:05:30
#SBATCH --job-name=gromacs
#SBATCH --error=job.16.%J.err
#SBATCH --output=job.16.%J.out
#SBATCH --partition=standard

cd /home/shweta/water-cut1.0_GMX50_bare/3072
module load compiler/intel/2018.5.274
module load apps/gromacs/5.1.4/cpu/intel_18.5

export I_MPI_DEBUG=5
export OMP_NUM_THREADS=1
mpirun -np 4 gmx_mpi grompp -f pme.mdp -c conf.gro -p topol.top

time mpirun -np $SLURM_NTASKS gmx_mpi mdrun -s topol.tpr) 2>&1 | tee
log_gromacs_40_50k_mpirun

```

Output Snippet:

```

Number of logical cores detected (48) does not match the number reported by
OpenMP (1).
Consider setting the launch configuration manually!
Running on 10 nodes with total 192 cores, 480 logical cores
  Cores per node:          0 - 48
  Logical cores per node:  48
Hardware detected on host cn072 (the node of MPI rank 0):
CPU info:
  Vendor: GenuineIntel
  Brand:  Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz
  SIMD instructions most likely to fit this hardware: AVX2_256
  SIMD instructions selected at GROMACS compile time: AVX2_256
Reading file /home/shweta/Gromacs/water-cut1.0_GMX50_bare/3072/topol.tpr,
VERSION 5.1.4 (single precision)
Changing nstlist from 10 to 20, rlist from 1 to 1.032
The number of OpenMP threads was set by environment variable
OMP_NUM_THREADS to 1 (and the command-line setting agreed with that)
NOTE: KMP_AFFINITY set, will turn off gmx mdrun internal affinity
      setting as the two can conflict and cause performance degradation.
      To keep using the gmx mdrun internal affinity setting, set the
      KMP_AFFINITY=disabled environment variable.
Overriding nsteps with value passed on the command line: 50000 steps, 100
ps
Will use 360 particle-particle and 120 PME only ranks
This is a guess, check the performance at the end of the log file
Using 480 MPI processes
Using 1 OpenMP thread per MPI process
Back Off! I just backed up ener.edr to ./#ener.edr.2#
starting mdrun 'Water'
50000 steps,      100.0 ps.

Average load imbalance: 5.5 %
Part of the total run time spent waiting due to load imbalance: 3.0 %
Average PME mesh/force load: 1.252

```

Part of the total run time spent waiting due to PP/PME imbalance: 13.2 %
NOTE: 13.2 % performance was lost because the PME ranks had more work to do than the PP ranks.
You might want to increase the number of PME ranks or increase the cut-off and the grid spacing.

	Core t (s)	Wall t (s)	(%)
Time:	204872.624	427.847	47884.5
	(ns/day)	(hour/ns)	
Performance:	20.195	1.188	

Acknowledging the National Supercomputing Mission in Publications

If you use supercomputers and services provided under the National Supercomputing Mission, Government of India, please let us know of any published results including Student Thesis, Conference Papers, Journal Papers and patents obtained.

Please acknowledge the National Supercomputing Mission as given below:

We acknowledge National Supercomputing Mission (NSM) for providing computing resources of ‘PARAM Ganga’ at IIT Roorkee, which is implemented by C-DAC and supported by the Ministry of Electronics and Information Technology (MeitY) and Department of Science and Technology (DST), Government of India.

Also, please submit the copies of dissertations, reports, reprints and URLs in which “National Supercomputing Mission, Government of India” is acknowledged to:

HoD HPC Technologies,
Centre for Development of Advanced Computing,
CDAC Innovation Park,
S.N. 34/B/1,
Panchavati, Pashan,
Pune – 411008, Maharashtra

Communication of your achievements using resources provided by National Supercomputing Mission, will help the Mission in measuring outcomes and gauging the future requirements. This will also help in further augmentation of resources at a given site of National Supercomputing Mission.

Getting Help – PARAM Ganga Support

We suggest that you please refer to these four easy steps to generate a Ticket related to the issue you are experiencing.

Your Ticket will be assisted by the GANGA Support team. The ticket generated will be closed only when the related issue gets resolved.

You can generate a new ticket for any of the new issue that you are experiencing.

Steps to Create a New Ticket

1. Place the URL (<https://paramganga.iitr.ac.in/support>) in your browser.
2. On the right-top corner of the page click **Sign In**. Refer to Fig: 38 for the same.

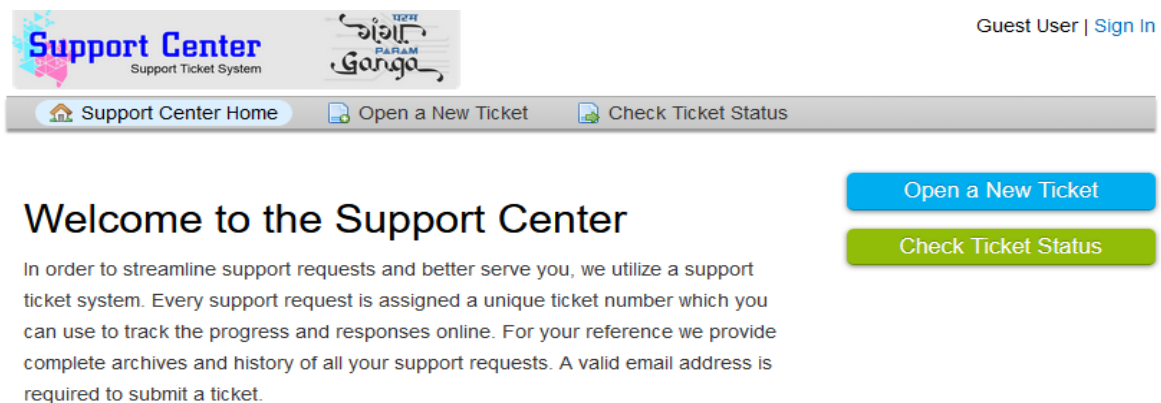
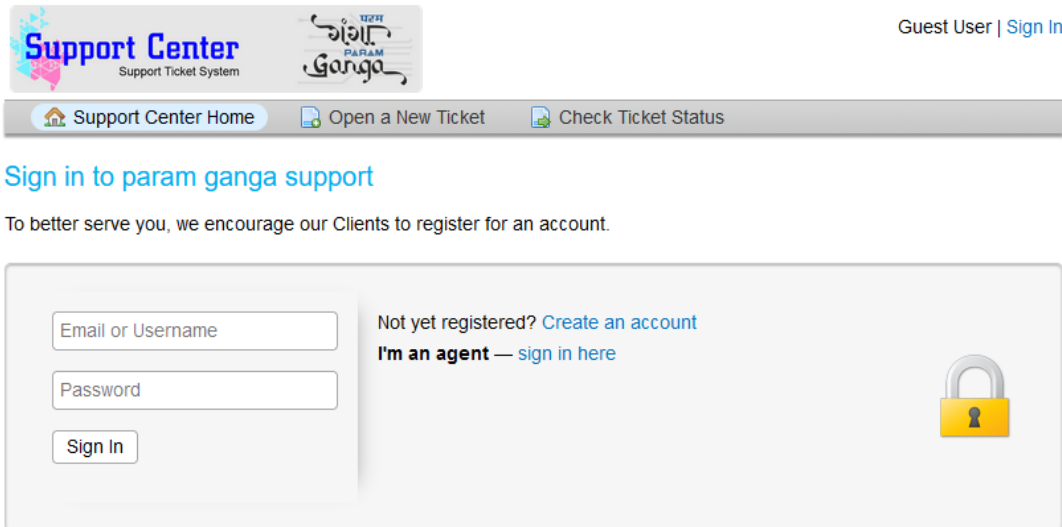


Figure 38 – Snapshot of Ticketing System

3. Sign in by using the Username and Password that you use for logging to the Cluster. Refer to Fig: 39 for the same.



If this is your first time contacting us or you've lost the ticket number, please [open a new ticket](#)

Figure 39 - Snapshot of Ticketing System

4. Select a **Help Topic** from the Dropdown and then click on **Create Ticket**. Refer to Fig: 40 for the same

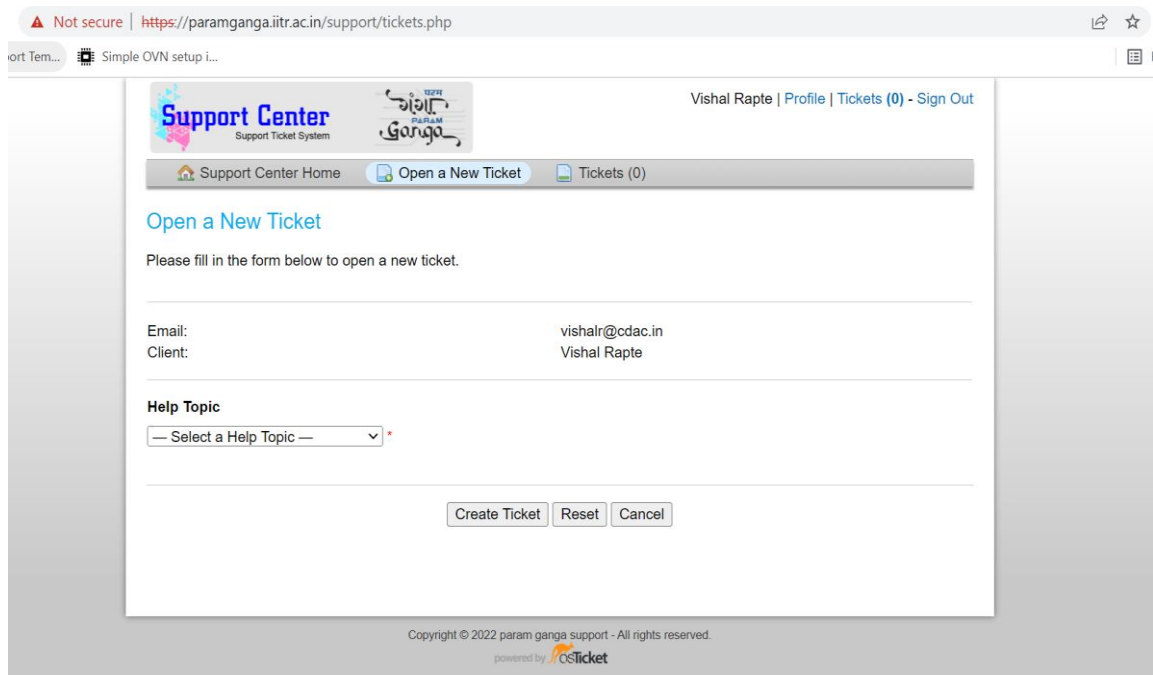
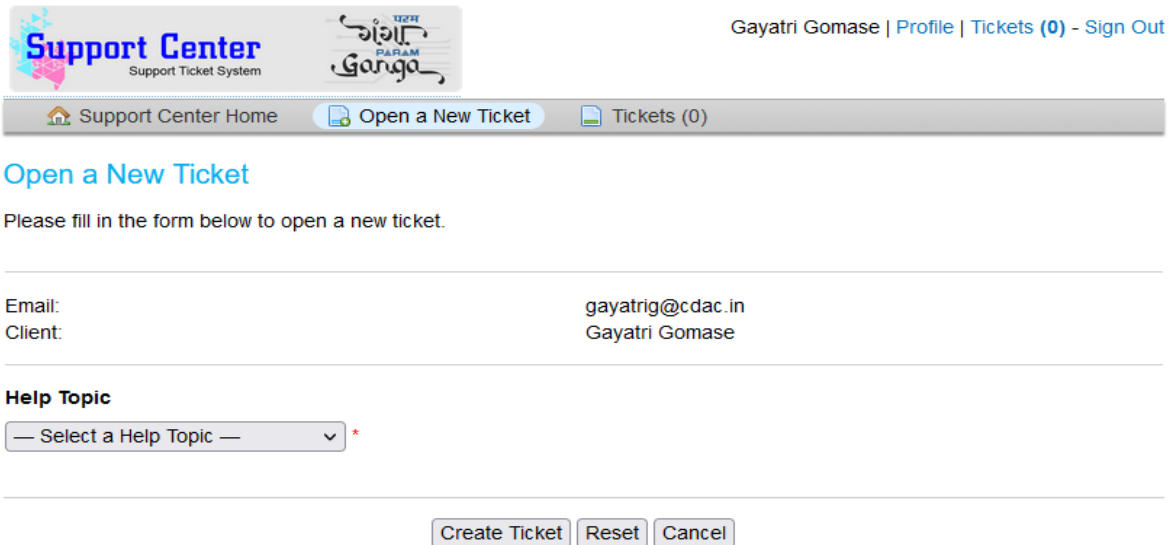


Figure 40 - Snapshot of Ticketing System

5. Please fill in the details of your issue in the fields given and then click on Create ticket.



The screenshot displays the 'Support Center' interface for the 'Support Ticket System'. The header includes the user's name 'Gayatri Gomase', a link to their 'Profile', a notification for 'Tickets (0)', and a 'Sign Out' option. Below the header is a navigation bar with three buttons: 'Support Center Home', 'Open a New Ticket' (which is highlighted), and 'Tickets (0)'. The main content area is titled 'Open a New Ticket' and contains the instruction: 'Please fill in the form below to open a new ticket.' The form fields are as follows:

Email:	gayatrig@cdac.in
Client:	Gayatri Gomase

Below the form fields is a section titled 'Help Topic' with a dropdown menu that currently shows '— Select a Help Topic —'. At the bottom of the form are three buttons: 'Create Ticket', 'Reset', and 'Cancel'.

Figure 41 - Snapshot of Ticketing System

Once the Ticket is generated, an acknowledgement e-mail will be sent to your official e-mail address. The e-mail will also contain the Ticket number along with reference to the ticket that you have generated.

In case of any difficulty while accessing GANGA Support you can reach us via e-mail at paramganga@iitr.ac.in

Closing Your Account on PARAM Ganga

When once you have completed your research work and you no longer need to use PARAM Ganga, you may please close your account on PARAM Ganga. Please raise a ticket by following the URL <https://paramganga.iitr.ac.in/support>. The system administrator will guide you about the “Closure Procedure”. You will need clearance from your project-coordinator/ Supervisor/ Head of the Department about you having surrendered this resource for getting “no dues” certificate from the institute.



PARAM Ganga ACCOUNT REQUEST FORM

User Details:

First Name: _____ Last Name: _____

Organization Name: _____

Organization Address: _____

Gender: _____

Department: _____

Designation: _____

(Designation: If student, provide the details below)

Roll No.: _____ Course: _____ Academic Year: _____

Official Email address: _____

Office no.: _____ Mobile no.: _____

(If Research, provide the details below)

Nature of the Research:

Project Details:

Project Name:

Nature of the Project:

Brief Description of the Project:

Project Start Date: _____ Project Duration: _____

Proposed work on PARAM Ganga & Requirement of resources:

PARAM Ganga HPC facility usage policies:

1. The resources provided to you on PARAM Ganga facility should not be used for any commercial purpose i.e., it is restricted for the academic use like research projects, academic projects, NSM projects, NSM approved MSME projects and scientific projects.
2. Sharing your login credentials with some third person will revoke the responsibility of PARAM Ganga administration committee for data theft and your account will also be disabled. The third person will also be held accountable for misusing the PARAM Ganga facility.
3. It is strictly recommended that you should not run jobs on login node and any such incident reported will result in cancellation of the job and any repeat action will result in closure of your account.
4. You will be responsible for informing the PARAM Ganga administration about your project completion, project cancellation and moving or copying data related to your project from PARAM Ganga.
5. You will be solely responsible for keeping your password strong and safe.

6. If found in any engagement or promotion of activities like hacking, reverse-engineering, violating intellectual property rights on or using the PARAM Ganga facility, you will be barred from having account on any supercomputer setup under the National Supercomputing Mission.
7. The facility is built with least downtime requirement; however, it depends on various factors like Hardware reliability, Power outage, network outage, scheduled maintenance due to which the facility could be unavailable completely/partially. Notification of all scheduled / unscheduled maintenance will be made known to the users via Website, Email, broadcast message, newsgroups etc.
8. This facility will not be used for any purpose connected with Chemical or Biological or nuclear weapons or missiles capable of delivering such Weapons.
9. Acknowledging the usage of the facility is mandatory.

If you use supercomputers and services provided under the National Supercomputing Mission, Government of India, please let us know of any published results including Student Thesis, Conference Papers, Journal Papers and patents obtained.

Performa for Acknowledging the usage:

Performa for Acknowledging the usage: We acknowledge National Supercomputing Mission (NSM) for providing computing resources of ‘PARAM Ganga’ at IIT Roorkee, which is implemented by C-DAC and supported by the Ministry of Electronics and Information Technology (MeitY) and Department of Science and Technology (DST), Government of India.

Also, please submit the copies of dissertations, reports, reprints and URLs in which “National Supercomputing Mission, Government of India” is acknowledged to:

HoD, HPC Technologies,
Centre for Development of Advanced Computing,
CDAC Innovation Park,
S.N. 34/B/1,

Panchavati, Pashan,
Pune – 411008
Maharashtra

Email: paramganga@iitr.ac.in

Communication of your achievements using resources provided by National Supercomputing Mission, will help the Mission in measuring outcomes and gauging the future requirements. This will also help in further augmentation of resources at a given site of National Supercomputing Mission.

I acknowledge the above-mentioned usage policies & terms and conditions.

User’s signature

Recommended/Not Recommended

Signature and seal of HoD/Head of Organization:

Name:

Designation:

Department:

Official Email address:

Only for Official Use

Approving Authority for NSM

Verified by:

Approving Authority:

Approved/Not Approved

Remarks:

Name, Signature and seal of approving authority

Information required for NSM (National Supercomputing Mission) users

Domain(s)*:

Sub-domain(s)*:

Application name(s)*:

(Indicative list of Domains and some of its applications)

Domain Name	Application Name
Astronomy & Astrophysics	ATHENA, CosmoMC
Atomic & Molecular Sciences	Gromacs, LAMMPS, NAMD, AMBER (Open Source)
Computational Biology	Biopython
Bioinformatics	mpiBlast, Clustaw- MPI,Fasta, Artemis, T-coffee
Chemical Sciences	Gromacs, LAMMPS, NAMD
Climate & Environment Sciences	MOM, Weather Research Forecasting model (WRF), COSMO
Computational Fluid Dynamics	OpenFoam, Tycho, Gerris flow Solver
Computational Physics	OOFEM
Computational Sciences	Gromacs, LAMMPS, NAMD, AMBER (open source)
Data analytics	RStudio, Apache Spark
Geological Sciences	Ferret
Data Visualization	GRADS, Ferret, ParaView

Material Sciences	Quantum Espresso, Q-chem
Quantum Mechanics	Abinit, NWChem, CP2K
Structural Engineering Mechanics	CODE-ASTER
AI/ML/DL	Tensorflow, Nvidia digits, pandas, numpy
Image Processing	OpenCV, Matplotlib, Scikit-image
Atmospheric/Ocean Modelling	MOM, Weather Research Forecasting model (WRF)

Please specify other application name if not listed above

(* form may get rejected if no mandatory information is provided)

References

1. <https://lammps.sandia.gov/>
2. <https://www.openacc.org/>
3. <https://www.openmp.org/>
4. <https://computing.llnl.gov/tutorials/mpi/>
5. <https://developer.nvidia.com/cuda-zone>
6. <https://www.mmm.ucar.edu/weather-research-and-forecasting-model>
7. <http://www.gromacs.org/>
8. <https://www.openfoam.com/>
9. <https://slurm.schedmd.com/>
10. https://www.tutorialspoint.com/gnu_debugger/what_is_gdb.htm
11. <https://nsmindia.in/>
12. https://en.wikipedia.org/wiki/Deep_learning
13. <https://docs.conda.io/en/latest/>
14. <https://docs.conda.io/en/latest/miniconda.html>
15. <https://www.tensorflow.org/>
16. <https://www.tensorflow.org/install>
17. <https://github.com/PaddlePaddle/Paddle>
18. <https://keras.io/>
19. <https://pytorch.org>
20. <https://mxnet.apache.org>
21. <https://software.intel.com/en-us/distribution-for-python>
22. <https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide>
23. <https://github.com/spack/spack>
24. https://spack.readthedocs.io/en/latest/getting_started.html
25. https://spack.readthedocs.io/en/latest/basic_usage.html
26. https://spack.readthedocs.io/en/latest/packaging_guide.html
27. https://spack.readthedocs.io/en/latest/build_systems.html
28. <https://spack.readthedocs.io/en/latest/>