PARAM Rudra

USER MANUAL

Last Updated: 17th May 2024

www.cdac.in

Copyright Notice

Trademarks

Intended Audience

This document is meant for PARAM Rudra 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.
Italics	Variables or placeholders or special terms in the document.
Console text	Console commands

Getting help

For technical assistance or license renewal, please send an email to rudrasupport@iuac.res.in.

Give us your feedback

We value your feedback. Kindly send your comments on content of this document to rudrasupport@iuac.res.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.

Table of Content

Introduction

This document is the user manual for the PARAM Rudra Supercomputing facility at SNBNCBS, S. N. Bose National Centre for Basic Sciences Kolkata. 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 the user's Laptop/ Desktop etc. In short, the manual describes all that one needs to know to effectively utilize PARAM Rudra.

The supercomputer PARAM Rudra is based on heterogeneous and hybrid configuration of Intel Xeon 2nd Gen Cascade Lake processors, and NVIDIA Ampere A100 GPU cards. The system was designed and implemented by HPC Technologies group, Centre for Development of Advanced Computing (C-DAC).

It consists of 4 Login nodes, 6 Management and 162 (CPU+GPU+HM) compute nodes with total peak computing capacity of (CPU+GPU+HM) 838 **PFLOPS.**

System Architecture and Configuration

System Hardware Specifications

PARAM Rudra system is based on the Intel Xeon Gold 6240R with a total peak performance of 838 PFLOPS. The cluster consists of compute nodes connected with the InfiniBand HDR100 Low-Latency, High-Bandwidth InfiniBand interconnect network. The system uses the Lustre parallel file system.

- Total number of Nodes: 172 (10 + 162)
  - Login Nodes: 4
  - Management Nodes: 6
  - CPU only Nodes: 96
  - GPU Nodes: 8
  - GPU ready Nodes: 26
  - High Memory CPU only Nodes: 32

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 Rudra 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: 4	
2* Intel Xeon G-6240R Cores = 48, 2.4 GHz	Total Cores = 192 cores
Memory= 192 GB each	Total Memory = 768 GB

Service Nodes

PARAM Rudra is an aggregation of a large number of nodes connected through networks. Management nodes play a crucial role in managing and monitoring every component of PARAM Rudra cluster. This includes monitoring the health, load, and utilization of individual components, as well as providing essential services such as security, management, and monitoring to ensure the cluster functions smoothly.

Management nodes: 6	
2* Intel Xeon G-6240R Cores = 48, 2.4 GHz	Total Cores = 288 cores
Memory= 192 GB	Total Memory= 1152 GB

CPU Compute Nodes

CPU nodes are the individual machines dedicated to performing computational tasks. These nodes collectively form the computational power of the cluster. 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.

CPU only Compute Nodes: 96	
2* Intel Xeon G-6240R Cores = 48, 2.4 GHz	Total Cores = 4608 cores
Memory= 192 GB, DDR4 2933 MHz	Total Memory=18,432 GB
SSD = 800 GB local per node	

GPU Ready Compute Nodes

GPU Ready Compute Nodes are similar to CPU Compute nodes which can be upgraded to support GPU computations in the future.

GPU Compute Nodes: 26	
2* Intel Xeon G-6240R Cores = 48, 2.4 GHz	Total Cores = 1248 cores

Memory= 192 GB, DDR4 2933 MHz	Total Memory= 4992 GB
SSD = 800 GB (local) per node	

GPU Compute Nodes

GPU Compute Nodes feature accelerators cards that offer significant acceleration for parallel computing tasks using frameworks like CUDA and OpenCL. By harnessing the computational power of modern GPUs, these nodes are utilized for tasks such as scientific simulations, deep learning, and data analytics, providing high computational power and memory.

GPU Compute Nodes: 8	
2* Intel Xeon G-6240R	Total Cores = 384 cores
Cores = 48, 2.4 GHz	
Memory = 192 GB, DDR4 2933 MHz	Total Memory = 1536 GB
SSD = 800 GB (local) per node	
2*Nvidia A100 per node	
GPU Cores per node= 2*6912= 13824	
GPU Memory = 80 GB HBM2e per Nvidia A100	

High Memory Compute Nodes

High Memory Compute nodes are specialized nodes designed to handle workloads that require a large amount of memory.

High memory Compute nodes: 32	
2* Intel Xeon G-6240R	Total Cores = 1536 cores
Cores = 48, 2.4 GHz	
Memory= 768 GB, DDR5 2933 MHz	Total Memory= 24576 GB
SSD = 800 GB (local) per node	

Storage

- Based on Lustre parallel file system
- Total useable capacity of 1.0 PiB Primary Storage
- Throughput 100 GB/s

PARAM Rudra Architecture Diagram

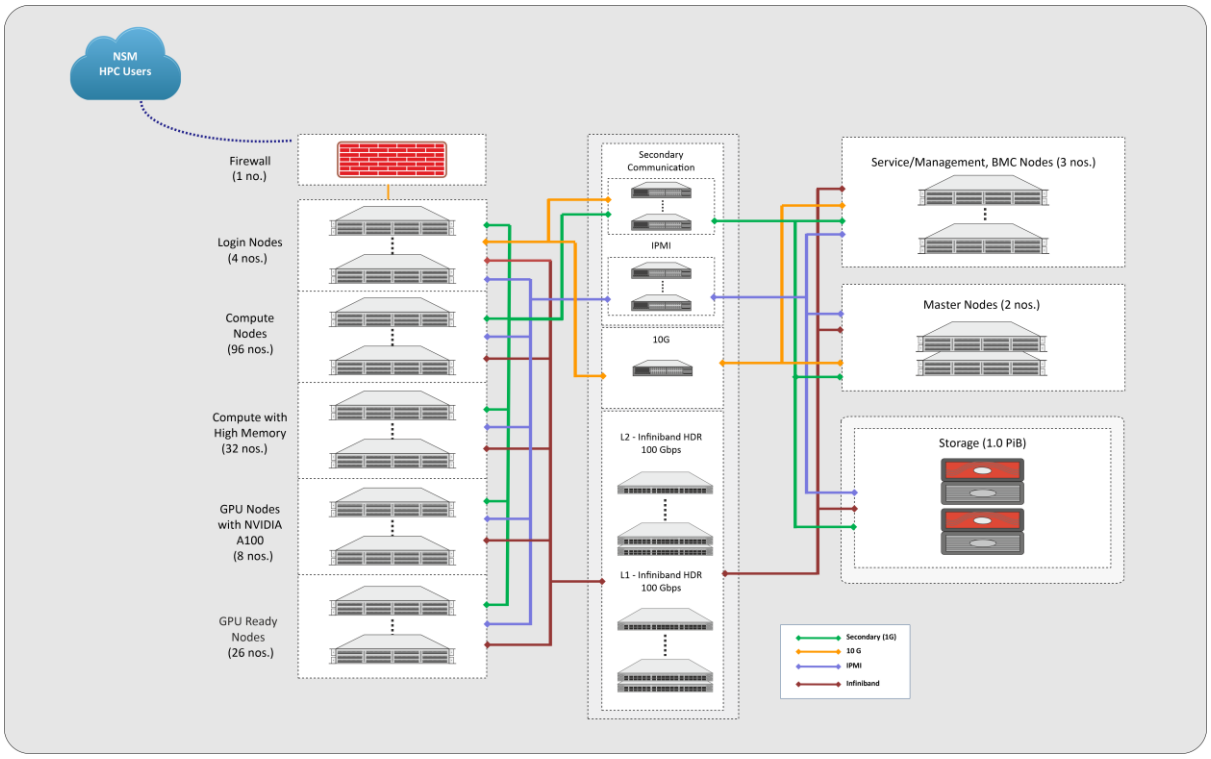


Figure 1 - PARAM Rudra Architecture Diagram

Operating System

The operating system on PARAM Rudra is Linux – Alma 8.9

Network infrastructure

A robust network infrastructure is essential for implementing the basic functionalities of a cluster. These functionalities include:

a) Management functionalities, such as monitoring, troubleshooting, starting and stopping various components of the cluster. The network/ or portion of the network that implements this functionality is referred to as the Management fabric.

b) Ensuring fast read/ writes access to the storage, the network or portion of the network that implements this functionality is referred to as the storage fabric.

c) Ensuring fast I/O operations, such as connecting to other clusters and connecting the cluster to various users on the campus LAN. The network or portion of the network that implements this functionality is referred to as the I/O Fabric.
d) Ensuring High-Bandwidth, Low-latency communication among processors is essential for achieving high-scalability. The network or portion of the network that implements this functionality is referred to as Message Passing Fabric.

Technically, all the above functionalities can be implemented in a single network. However, for optimal performance, economic suitability, and meeting specific requirements, these functionalities are implemented using two different networks based on different technologies, as explained below:

Primary Interconnection Network

InfiniBand: HDR 100 Gbps

Computing nodes of PARAM Rudra are interconnected by a high-bandwidth, low-latency interconnect network, specifically InfiniBand: HDR 100 Gbps. InfiniBand, a high-performance communication architecture owned by Mellanox, 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 most commonly available interconnection network. No additional modules or libraries are required for Gigabit Ethernet. Both Open MPI, MPICH implementations will work over Gigabit Ethernet.

Software Stack

Software Stack is an aggregation of software components that work together to accomplish various task. These tasks can range from facilitating users in executing their jobs to enable system administrator to manage the system efficiently. Each software component within the stack is equipped with the necessary tools to achieve its specific task, and there may be multiple components of different flavors for different sub-tasks. Users have the flexibility to mix and match these components according to their preferences. For users, the primary focus is on preparing executables, executing them with their datasets, and visualizing the output. This typically involves compiling codes, linking them with communication libraries, math libraries, and numerical algorithm libraries, preparing executables, running them with desired datasets, monitoring job progress, collecting results, and visualizing output.

System administrators, on the other hand, are concerned with ensuring optimal resource utilization. To achieve this, they may require installation tools, health-check tools for all components, efficient schedulers, and tools for resource allocation and usage monitoring.

The software stack provided with this system have a wide range of software components that meet the needs of both users and administrators. Figure 2 illustrates the components of the software stack.

C-CHAKSHU, a multi-cluster management tool designed to help administrator operate the HPC facility efficiently. It also enables the users 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:
https://paramrudra.bose.res.in/chakshu-front

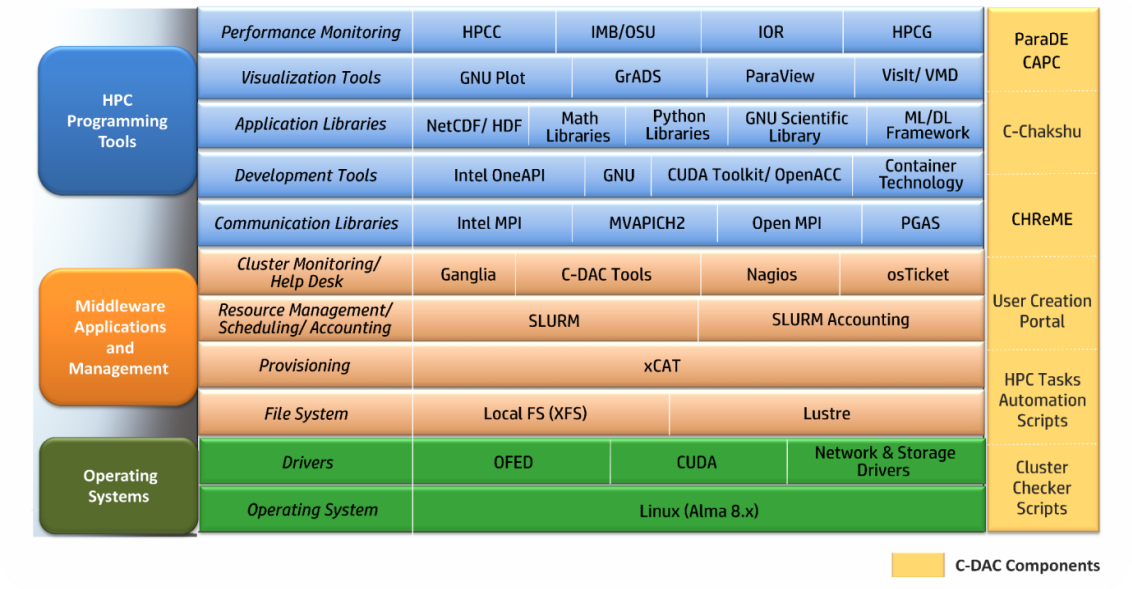


Figure 2- Software Stack

Functional Areas	Components
Base OS	Alma 8.9
Architecture	X86_64
Provisioning and Cluster Manager	xCAT 2.16.5
Monitoring Tools	C-CHAKSHU, Nagios, Ganglia
Resource Manager	SLURM- 23.01.1

I/O Services	Lustre Client
High Speed Interconnects	Mellanox InfiniBand (MLNX_OFED_LINUX-23.10)
Compiler Families	GNU (gcc, g++, GNU Fortran) Intel Compiler (icc, ifort, icpc)
MPI Families	MVAPICH, Open MPI, MPICH

First Things First

Getting an Account on PARAM Rudra

To begin with, you need to get an account on PARAM Rudra.

Recommended process for creating a user account to access the PARAM Rudra:

1. Visit nsmindia.in
2. Navigate to the "How to access NSM HPC System" section, where you will find a link to the User Creation portal.
3. Click on the provided link to access the registration page.
4. Fill in all required information on the registration page.
5. Select SNBNCBS, S. N. Bose National Centre for Basic Sciences Kolkata as the institute.
6. Upload the necessary documents as instructed.
7. Once the form is complete, submit the details.
8. The NSM committee will review the submission.
9. If accepted, users will receive an email containing their user credentials and allocated cluster.

How to access the cluster

To access cluster using Windows:

To access PARAM Rudra, there are few tools available, please see some below:

1. PuTTY is the most popular open source ssh client application for Windows. Following are the steps:
   a) Download PuTTY from its official website.
   b) Install PuTTY on your computer.
   c) Launch Putty from your desktop or Start menu.
   d) In the dialog, locate the "Hostname or IP Address" input field.
   e) Enter the hostname of the cluster: paramrudra.bose.res.in
   f) For SNBNCBS users, use port 22
   g) For external users, use port 4422
   h) Select open, then enter your username
   i) Enter the captcha when prompted, then input your password.
   j) Press Enter to proceed with the connection.

2. Another popular tool is MobaXterm, which is a third party freely available tool which can be used to access the HPC system and transfer files to the PARAM Rudra system through your local systems (laptop/desktop). Here are the steps:
   a) Download MobaXterm from its official website.
   b) Install MobaXterm on your computer.
   c) Launch MobaXterm from your desktop or Start menu.
   d) Click on the "Session" button in MobaXterm.
   e) Enter the hostname, along with your username.
   f) For SNBNCBS users, use port 22
   g) For external users, use port 4422
   h) Enter the captcha when prompted, then input your password.
   i) Press Enter to proceed with the connection.

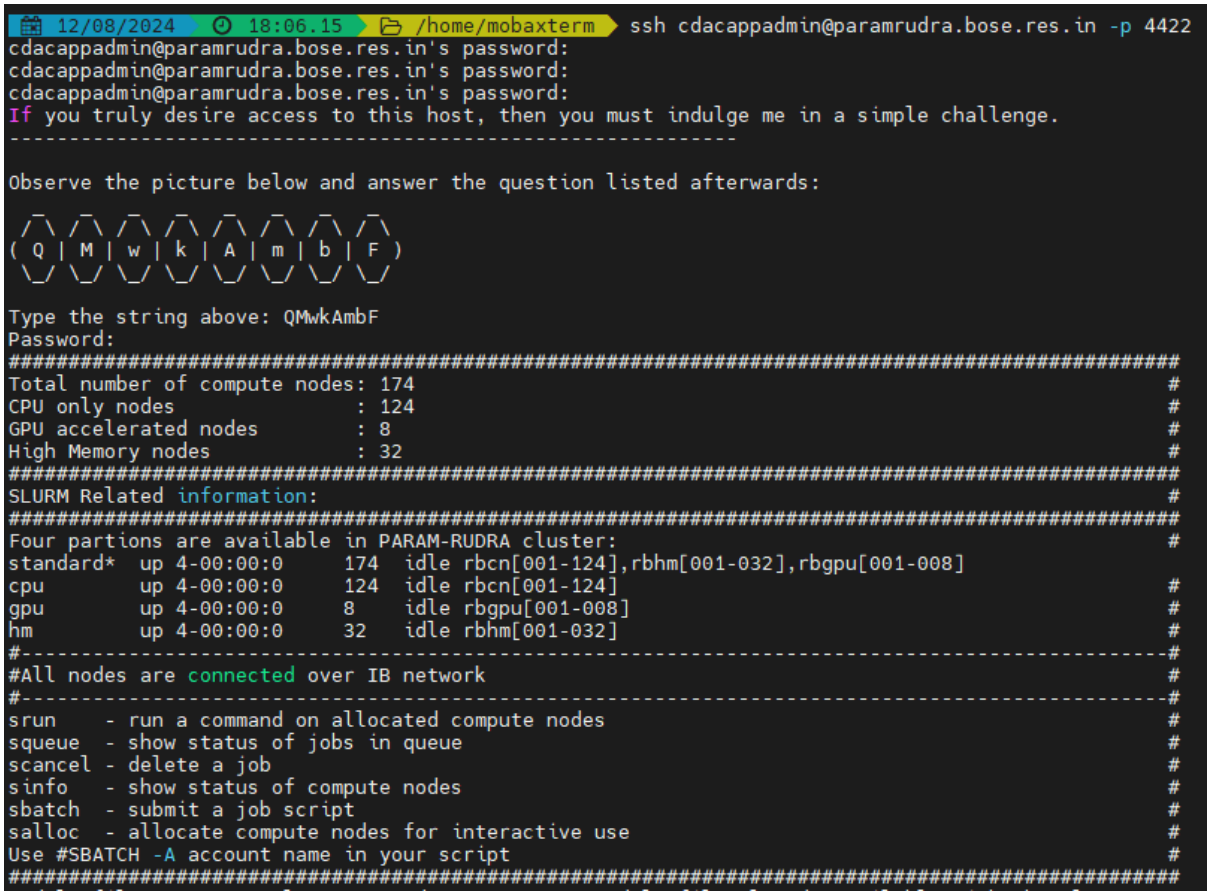


Figure 3 - A snapshot of command using MobaXterm

1. Command Prompt (Windows native application)

This is a native tool for Windows machines which can be used to transfer data from the PARAM Rudra system through your local systems (laptop/desktop).

```
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin> ssh cdacappadmin@paramrudra.bose.res.in -p 4422
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:

  _   _   _   _   _   _   _   _
 / \ / \ / \ / \ / \ / \ / \ / \
( p | Z | G | Q | A | K | F | Q )
 \_/ \_/ \_/ \_/ \_/ \_/ \_/ \_/

Type the string above: pZGQAKFQ
Password:
###############################################################################################################
Total number of compute nodes: 174                                                                            #
CPU only nodes               : 124                                                                            #
GPU accelerated nodes        : 8                                                                              #
High Memory nodes            : 32                                                                             #
###############################################################################################################
SLURM Related information:                                                                                    #
###############################################################################################################
Four partions are available in PARAM-RUDRA cluster:                                                           #
standard*  up 4-00:00:0     174  idle rbcn[001-124],rbhm[001-032],rbgpu[001-008]
cpu        up 4-00:00:0     124  idle rbcn[001-124]                                                          #
gpu        up 4-00:00:0     8    idle rbgpu[001-008]                                                         #
hm         up 4-00:00:0     32   idle rbhm[001-032]                                                          #
#-------------------------------------------------------------------------------------------------------------#
#All nodes are connected over IB network                                                                      #
#-------------------------------------------------------------------------------------------------------------#
srun    - run a command on allocated compute nodes                                                            #
squeue  - show status of jobs in queue                                                                        #
scancel - delete a job                                                                                        #
sinfo   - show status of compute nodes                                                                        #
sbatch  - submit a job script                                                                                 #
salloc  - allocate compute nodes for interactive use                                                          #
Use #SBATCH -A account name in your script                                                                    #
###############################################################################################################
```

Figure 4 - A snapshot of the "scp" command using Windows command prompt.

2. PowerShell (Windows native application)

This is a native tool for Windows machines which could be used to transfer data from the PARAM Rudra system through your local systems (laptop/desktop).

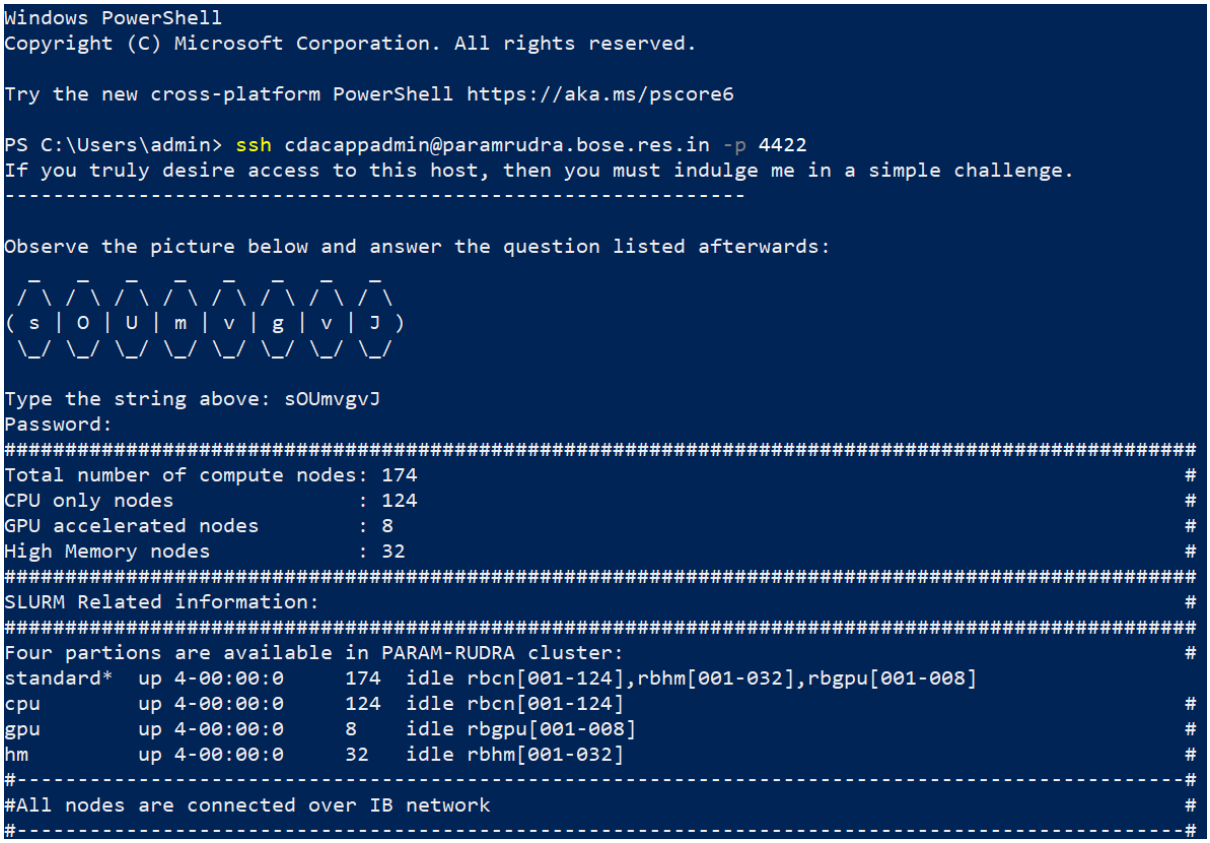


Figure 5 - A snapshot of the "scp" command using Windows PowerShell.

To access cluster using Mac or Linux

Both Mac and Linux systems provide a built-in SSH client, eliminating the need to install any additional package. To connect to a SSH server, open the terminal and type the following command:

ssh[username]@[hostname]

For example, to connect to PARAM Rudra cluster:
For **IUAC** Users:x

```
ssh user1@paramrudra.bose.res.in
```

For External Users:

```
ssh user1@paramrudra.bose.res.in -p 4422
```

After entering captcha, you will be prompted for a password. Once entered, you will be connected to the server.

After getting credentials you may access the cluster, please remember the following points:

- When you log in to the cluster, you will land on the login nodes. The login node serves as the primary gateway to the rest of the cluster, housing a job scheduler (known called Slurm) and other applications for creating and submitting the job. You can submit jobs to the queue, and they will execute when the required resources become available.
- Please refrain from running jobs directly on the login node. Login nodes are intended for compiling codes, transferring data and submitting jobs. If you run your job directly on the login node, it will be terminated.
- By default, two directories are available (i.e. /home and /scratch). These directories are available on the 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. Users are requested to regularly back up their own data in scratch directory. As per policy, any files not accessed in the last three months will be permanently deleted.

First login

Whenever a newly created user on PARAM Rudra attempts to log in with the user ID and temporary password provided via email by PARAM Rudra support, it is mandatory for the user to change the password to one of their choosing. This ensures the security of your account. It is recommended to use a strong password containing a combination of lowercase and uppercase letters, numbers, and a few special characters that are easy for you to remember.

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?

1. Please open a ticket regarding this issue, and the support team will assist you with your problem. Follow the steps below:
2. Visit the PARAM Rudra support site, which is the ticketing tool, by clicking on the following link: PARAM Rudra Support.
3. Log in using your username or registered email ID.
4. Raise a ticket to request a password reset.
5. The support team will respond with an email for verification.
6. Once you acknowledge the email, the password will be reset for you, and you will receive an email confirming the same.
7. You can then log in using the temporary password provided and set a new password of your choice.

How to change the password:

Use the passwd command to change your user password. Enter your current password, followed by your new password, and then confirm the new password.

Transferring files between local machine and HPC cluster

Users need to have their data and applications related to their project or research work on PARAM Rudra. To store the data, special directories named “home” have been made available to the users. While these directories are common to all the users, each user will have their own directory with their username in the “/home/” directory, where they can store their data.

/home/<username>/ : This directory is generally used by the user to store their data and if need install their own applications.

However, there is a limit to the storage provided to users. The limits have been defined according to quota over these directories, and 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 the ‘Windows’ operating system will have access to methods and tools native to Microsoft Windows, as well as tools that can be installed on their Windows machine. Linux operating system users, however, do not require any tool. They can simply use the “scp” command on their terminal. Here’s how:

```
scp -r -P 22 <path to the local data directory>
<username>@paramrudra.bose.res.in:<path to directory on HPC where to save
the data>
```

Note: use port 22 within SNBNCBS Institute

Example:

Same Command could be used to transfer data from the HPC system to other HPC system, or your own system.

```
scp -r -P 4422 <file path> <username@<cluster
IP/hostname>:/home/user/<path>
```

Note: use port 4422 for your system.

Note: The local system (laptop/desktop) must be connected to a network that allows access to the HPC system. Additionally, please ensure that the firewall settings on your laptop are configured to allow access from the HPC system.

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

Tools

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

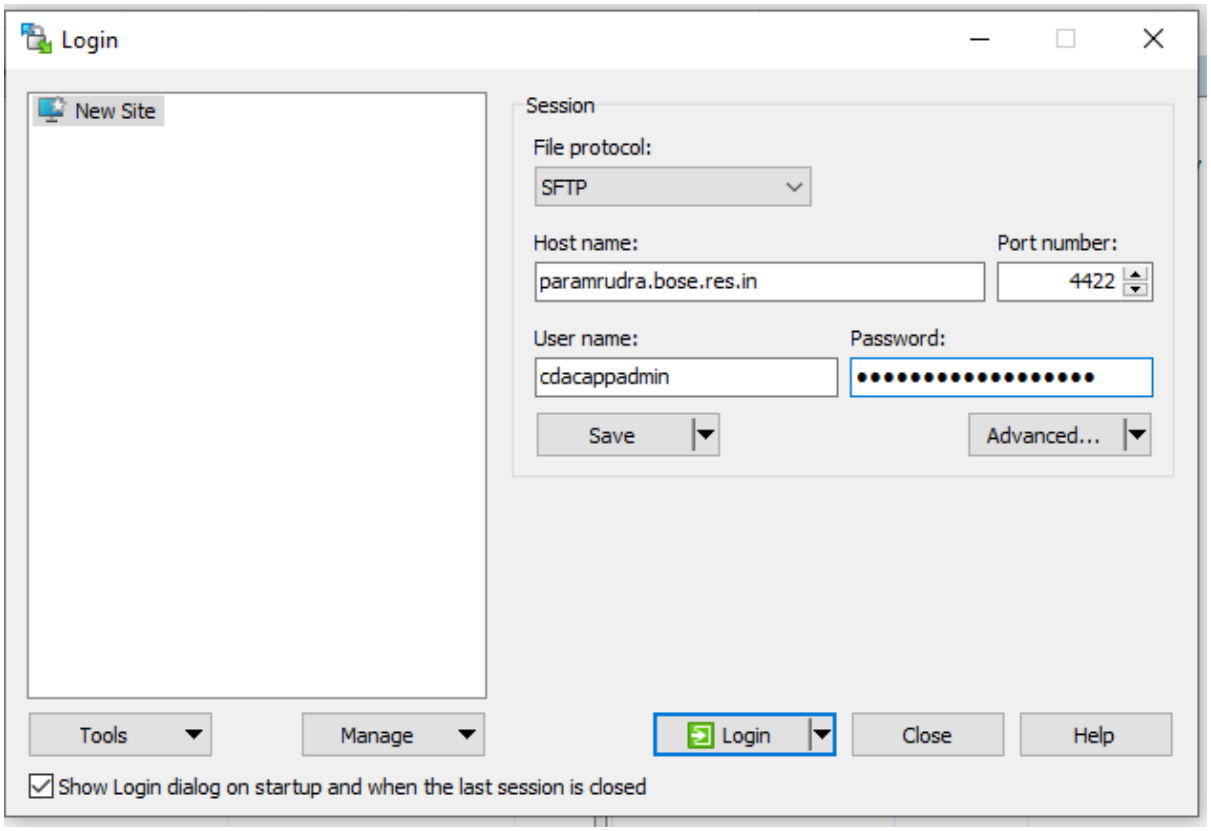


Figure 6 - A snapshot of the "WinSCP" tool to transfer file to and from remote computer.

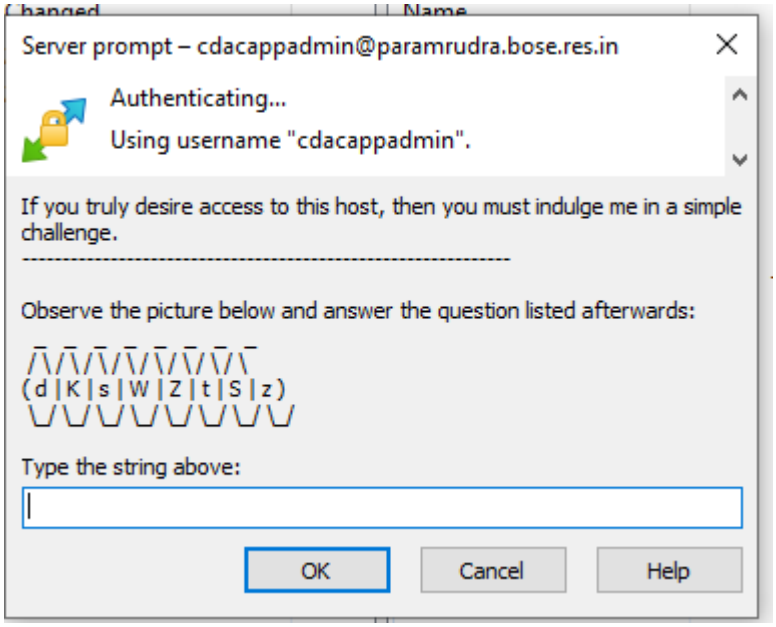


Figure 7–Enter Captcha/String

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

Resource Management

This section explains how you interact with the resource manager. It covers information about the resource manager, the definition of nodes within partitions, job policies, scheduler information, the process of submitting jobs to the cluster, monitoring active jobs and getting useful information about resource usage.

A cluster is a group of computers that work together to solve complex computational tasks and presents itself to the user as a single system. For the resources of a cluster (e.g. CPUs, GPUs, memory) to be used efficiently, a resource manager (also called workload manager or batch-queuing system) is important. While there are many different resource managers available, the resource manager at PARAM Rudra is SLURM. After submitting a job to the cluster, SLURM will try to fulfill the job's resource request by allocating resources to the job. If the requested resources are already available, the job can start immediately. Otherwise, the start of the job is delayed (pending) until enough resources are available. SLURM allows you to monitor active (pending, running) jobs and to retrieve statistics about finished jobs.

SLURM, which is open-source workload manager, efficiently allocates computing resources such as CPUs, GPUs, and memory to users' jobs, ensuring optimal resource utilization and job scheduling. SLURM provides features for job submission, monitoring, and management, allowing users to specify job requirements and dependencies. Slurm is a widely used batch scheduler in the top500 HPC list.

SLURM Partitions

Partition is a logical grouping of nodes that share similar characteristics or resources. Partitions are helpful to manage and allocate resources efficiently based on the specific requirements of jobs or users. PARAM Rudra consists of three types of computational nodes: i.e. CPU only nodes, High memory (with 768 GB memory) nodes and GPU-enabled GPGPU nodes.

The following partitions/queues have been defined to meet different user requirements:

1. **standard**: By default, all user job will be submitted to the standard partition which contains 154 nodes. These nodes consist of CPU and High Memory (HM) nodes.
2. **CPU**: This partition is specifically designed for nodes that only have CPU resources.
3. **GPU**: The GPU partition includes nodes equipped with NVIDIA A100 GPUs. Jobs submitted to this partition will run on nodes that can leverage the high-performance computing capabilities of A100 GPU cards for parallel processing tasks. The GPU

partition exclusively contains GPU nodes. If a user's wishes to submit a job only on GPU nodes, they need to specify the number of GPU cards with the partition name.

4. **hm:** The High Memory partition is intended for nodes with a substantial amount of RAM. Specifically, it accommodates CPU nodes that are equipped with 768 GB of RAM, allowing jobs requiring large memory resources to be executed efficiently.

QoS Job policy

Users have the flexibility to run up to 10 simultaneous jobs. They can run an 8-node job for 4 days, a 16-node job for 2 days, or a 32-node job for 1 day. The default policy of the cluster allows for a maximum wall time of 4 days per job. However, this policy can be tailored to individual user needs or adjusted for all users in the future, depending on cluster usage. Users will be informed about any changes made to the SLURM policy**.**

Walltime

The walltime parameter defines how long your job will run, with the maximum runtime determined by the QoS Policy. The default walltime for every job is 2 hours, so users are requested to explicitly specify the walltime in their scripts. If more than 4 days are required, users can raise a query on the support portal of PARAM Rudra, and it will be addressed on a case-by-case basis. If a job exceeds the specified walltime in the script, it will be terminated. Specifying the appropriate walltime improves scheduling efficiency, resulting in enhanced throughput for all jobs, including yours.

Scheduling Type

PARAM Rudra 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 =
      site_factor +
      (PriorityWeightAge) * (age_factor) +
      (PriorityWeightAssoc) * (assoc_factor) +
      (PriorityWeightFairshare) * (fair-share_factor) +
      (PriorityWeightJobSize) * (job_size_factor) +
      (PriorityWeightPartition) * (priority_job_factor) +
      (PriorityWeightQOS) * (QOS_factor) +
      SUM(TRES_weight_cpu * TRES_factor_cpu,
```

```
          TRES_weight_<type> * TRES_factor_<type>,
          ...)
      - nice_factor
```

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 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.

Association Factor: Each association can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request this association. This priority value is normalized to the highest priority of all the association to become the association factor.

Job Size Factor: The job size factor correlates to the number of nodes or CPUs the job has requested.

Nice Factor: Users can adjust the priority of their own jobs by setting the nice value on their jobs. Like the system nice, positive values negatively impact a job's priority and negative values increase a job's priority. Only privileged users can specify a negative value.

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.

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.

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.

Job Submission

We can submit jobs either through a SLURM script or by using the interactive method. Creating a SLURM script is the optimal way to submit a job to the cluster.

Submitting Batch Scripts Jobs

Here is the example of sample slurm script:

```
#!/bin/bash#!/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 package //
spack load intel-oneapi-compilers

cd  <Path of the executable>
a.out   (Name of the executable)
```

We can consider four cases of submitting a job here:

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 the pre-processing script.

Here's a simple job script.

Note that the Slurm -J option is used to give the job a name.

```
#!/bin/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 rbcn001
```

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 rbcn001
```

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 given reservation name 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 9 – Snapshot depicting the usage of "Job Array"

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

spack load intel-oneapi-compilers
cd /home/guest/Rajneesh/Rajneesh    #change to your required directory
export OMP_NUM_THREADS=${SLURM_ARRAY_TASK_ID}
/home/guest/Rajneesh/Rajneesh/md_omp
```

Running Interactive Jobs

Another way to run your job is interactively. You can run an interactive job as follows:

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

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

The command prompt of the allocated compute node will appear as soon as the job starts.

Exit the bash shell to end the job.

If the job is waiting for the resources, then this is how it will look :

```
$ job 1040 queued and waiting for resources
```

If after a while, it will allocate resources, then it will look like this:

```
$ job 1040 has been allocated resources
```

If you exceed the time or memory limit the job will also abort.

Please note that PARAM Rudra is NOT meant for executing interactive jobs. However, it can be utilized to quickly verify the successful execution of a job before submitting a larger batch job with a high iteration count. It can also be used for running small jobs. However, it's important to consider that other users may also be utilizing this node, so it's advisable not to inconvenience them by running large jobs.

There are various use cases for requesting interactive resources, such as debugging (launching a job, adjusting setup parameters like compile options, relaunching the job, and making further adjustments) and interactive interfaces (inspecting a node, etc.).

Parameters used in SLURM job script

The job flags are used with the 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.

	Flag Syntax	Description
partition	--partition=<partition name>	Partition is a queue for the 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 the number of cores on the compute node.
resource feature	--gres=gpu:2	Request use of GPUs on the gpu compute nodes
account	--account=<group-slurm-account>	User may belong to multiple accounts. If only one account is allocated, it will be set as the default.
job name	--job-name="lammps"	Name of the job.
error file	--error=<filename_pattern>	Instruct Slurm to connect the batch script's standard error directly to the file name specified in the "filename pattern". By default, both standard output and standard error are directed to the same file.
output file	--output=<filename_pattern>	Instruct Slurm to connect the batch script's standard output directly to the file name specified in the "filename pattern". By default, both standard output and standard error are directed to the same file.

node list	-w, --nodelist	Request a specific list of hosts.
mail-type	--mail-type=	Notify users by email when certain event types occur. Valid 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), and ARRAY_TASKS (send emails for each array task). Multiple type values may be specified in a comma separated list
mail-user	--mail-user=<user email>	User to receive email notification of state changes as defined by --mail-type.
Reservation	--reservation=<reservation>	Allocate resources for the job from the named reservation.
Validate script	--test-only	Validate the batch script and return an estimate of when a job would be scheduled to run given the current job queue and all the other arguments specifying the job requirements. No job is actually submitted.
exclusive access to nodes	--exclusive	Exclusive access to compute nodes. The job allocation cannot share nodes with other running jobs

Sample SLURM Scripts for reference

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 package //
spack load intel-oneapi-compilers

cd  <Path of the executable>
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 core 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=cpu       // specifies partition name


spack load intel-oneapi-compilers  // To load the package

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
the 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
spack load intel-oneapi-compilers          // To load the package


// 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    //change to required path where command needs to be
executed
or
cd /home/manjuv/LAMMPS_2018COMPILER/lammps-22Aug18/bench

// Example 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

spack load intel-oneapi-compilers       // To load the package
//change to script submission directory
cd $SLURM_SUBMIT_DIR

// 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)

// Example Command to run the lammps in Parallel //

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

Listing Partition

sinfo displays information about nodes and partition allowing users to view available nodes in the partition within the cluster.

```
[cdacappadmin@login02 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
standard*    up 4-00:00:00      1  inval rbcn078
standard*    up 4-00:00:00     16   unk* rbcn[009-010,043-048,068,074,107-108,123-124],rbhm[031-032]
standard*    up 4-00:00:00    137   resv rbcn[001-007,011-036,038-042,049-067,069-073,075-077,079-106,109-122],rbhm[001-030]
standard*    up 4-00:00:00      2   down rbcn[008,037]
cpu          up 4-00:00:00      1  inval rbcn078
cpu          up 4-00:00:00     14   unk* rbcn[009-010,043-048,068,074,107-108,123-124]
cpu          up 4-00:00:00    107   resv rbcn[001-007,011-036,038-042,049-067,069-073,075-077,079-106,109-122]
cpu          up 4-00:00:00      2   down rbcn[008,037]
hm           up 4-00:00:00      2   unk* rbhm[031-032]
hm           up 4-00:00:00     30   resv rbhm[001-030]
gpu          up 4-00:00:00      1  down* rbgpu003
gpu          up 4-00:00:00      7   resv rbgpu[001-002,004-008]
```

Figure 8- Output of sinfo command

Monitoring jobs

Monitoring jobs on SLURM can be done using the command **squeue**. The command squeue provides high-level information about jobs in the Slurm scheduling queue (state information, allocated resources, runtime, etc .

```
$ squeue
```

```
[cdacappadmin@login02 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               163  standard hostname cdacappa PD       0:00      1 (ReqNodeNotAvail, UnavailableNodes:rbcn[008-0
```

The command scontrol provides even more detailed information about jobs and job steps.

It will report more detailed information about nodes, partitions, jobs, job steps, and configuration.

```
$ scontrol show job <jobid>
```

```
-032])
[cdacappadmin@login02 ~]$ scontrol show job 163
JobId=163 JobName=hostname
   UserId=cdacappadmin(60012) GroupId=cdacappadmin(60012) MCS_label=N/A
   Priority=1 Nice=0 Account=cdac QOS=normal
   JobState=CANCELLED Reason=ReqNodeNotAvail,_UnavailableNodes:rbcn[008-010,037,043-048,0
   Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=02:00:00 TimeMin=N/A
   SubmitTime=2024-08-12T18:59:48 EligibleTime=2024-08-12T18:59:48
   AccrueTime=2024-08-12T18:59:48
   StartTime=Unknown EndTime=2024-08-12T19:03:29 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2024-08-12T19:03:09 Scheduler=Main
   Partition=standard AllocNode:Sid=boson01:2903550
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=
   NumNodes=1 NumCPUs=1 NumTasks=N/A CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   ReqTRES=cpu=1,mem=1M,node=1,billing=1
   AllocTRES=(null)
   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=hostname
   WorkDir=/home/cdacappadmin
   Power=
```

Figure 12 – scontrol show job displays specific job information

scontrol update job <jobid>- set <new attribute value>

The above command change attributes of submitted job. Like time limit, nodelist, number of nodes, etc. For example:

```
scontrol update jobid=163 set TimeLimit=4-00:00:00
```

Deleting jobs:

Use the scancel command to delete active jobs. Users can cancel their own jobs only.

$ scancel <jobid>

```
$ scancel 163
$ squeue --me
  JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
```

Holding a job:

Use the scontrol command to hold the job.

$scontrol hold <jobid>

```
$ squeue
  JOBID PARTITION  NAME    USER   ST      TIME  NODES NODELIST(REASON)
    163 standard   simple  user1  PD      0:00      1 (Dependency)
138 standard   simple  user1   R      0:16      1  rbcn001

$ scontrol hold 163
$ squeue
  JOBID PARTITION  NAME    USER   ST     TIME  NODES NODELIST(REASON)
    163 standard   simple  user1  PD     0:00      1 (JobHeldUser)
    138 standard   simple  user1   R     0:32      1 rbcn001
```

Releasing a job:

```
$ scontrol release 139
$ squeue
  JOBID PARTITION  NAME     USER  ST       TIME  NODES NODELIST(REASON)
    163 standard   simple  user1  PD       0:00      1 (Dependency)
    138 standard   simple  user1   R       0:46      1 rbcn001
```

Getting Node and Partition details

scontrol show node <node name> - shows detailed information about compute nodes.

```
[cdacappadmin@login02 ~]$ scontrol show node rbcn002
NodeName=rbcn002 Arch=x86_64 CoresPerSocket=24
   CPUAlloc=0 CPUEfctv=48 CPUTot=48 CPULoad=48.13
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=rbcn002 NodeHostName=rbcn002 Version=23.11.8
   OS=Linux 4.18.0-513.5.1.el8_9.x86_64 #1 SMP Mon Nov 20 08:56:15 EST 2023
   RealMemory=1 AllocMem=0 FreeMem=53528 Sockets=2 Boards=1
   State=IDLE+RESERVED ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=standard,cpu
   BootTime=2024-08-05T12:55:59 SlurmdStartTime=2024-08-08T18:48:58
   LastBusyTime=2024-08-09T14:10:40 ResumeAfterTime=None
   CfgTRES=cpu=48,mem=1M,billing=48
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/a ExtSensorsWatts=0 ExtSensorsTemp=n/a
   ReservationName=Benchmark
```

Figure 10 – scontrol show node displays compute node information

scontrol show partition <partition name>- shows detailed information about a specific partition

```
[cdacappadmin@login02 ~]$ scontrol show partition hm
PartitionName=hm
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=02:00:00 DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=4-00:00:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED MaxCPUsPerSocket=UNLIMITED
   Nodes=rbhm[001-032]
   PriorityJobFactor=1000 PriorityTier=1000 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=1536 TotalNodes=32 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
   TRES=cpu=1536,mem=32M,node=32,billing=1536
```

Figure 11 – scontrol show partition displays specific partition details

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 Banking System, where each department can be allocated with some predefined 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 half yearly. Depending upon the policy, users will be informed when their account is created about how much CPU hours have been allocated to them.

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.

$ sacct -j <jobid>

Investigating a job failure

Job executions aren't always successful. There are various reasons for a job to stop or crash. The most common causes are:

- Exceeding resource limits
- Software-specific errors

This section discusses methods to gather information and find ways to avoid common issues.

It is important to collect error and output messages by either writing this information to the default location or specifying specific locations using the --error/--output option. Redirecting the error/output stream to /dev/null should be avoided unless you fully understand its implications, as error and output messages serve as the initial point for investigating job failures.

Exceeding Resource Limits

Each partition defines default and maximum time limits of the job runtime and memory usage. Within the job script, the current limits can be defined within the ranges. For better scheduling, the job requirements should be estimated and the limits should be adapted to the needs. Lower limits enable SLURM to find suitable scheduling opportunities more effectively. Additionally, specifying minimal resource overhead minimizes resource wastage.

If a job exceeds the runtime or memory limit, it will get killed by SLURM.

Software Errors

The exit code of a job is captured by Slurm and saved as part of the job record. For sbatch jobs the exit code of the batch script is captured. For srun, the exit code will be the return value of the executed command. Any non-zero exit code is considered a job failure, and results in job state of FAILED. When a signal was responsible for a job/step termination, the signal number will also be captured, and displayed after the exit code (separated by a colon).

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	#BATCH
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]

Addressing Basic Security Concerns

- Your account on PARAM Rudra is 'private to you'. You are responsible for any actions emanating from your account. It is suggested that you should never share the password with anyone.
- Do not grant permission of your home directory to any other user, as it may expose your personal files to unauthorized access.

Per user

- Every user will have quota of 50 GB of soft limit and X GB of hard limit with grace period of X days in HOME file system (/home) and X GB of soft limit and X GB of hard limit with grace period of X 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

It is important to note:

- Compilations are performed on the login node. Only the execution is scheduled via SLURM on the compute nodes.
- It is important to collect error/output messages, either by writing such information to the default location or by specifying specific locations using the --error or --output option. Error and output messages serves as the starting point for investigating a job failures. If not specified, the Job Id is also appended to the output and error filenames.
- Submitting a series of jobs (a collection of similar jobs) as array jobs instead of one by one is crucial for improving backfilling performance and thus job throughput, instead of submitting the same job repeatedly.
- 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

Loading modules through SPACK

PARAM Rudra extensively uses spack. The purpose of spack is to provide freedom to users for loading required applications or packages of specific versions with all its dependencies in the user environment. Users can find the list of all installed packages with their specific versions and dependencies. This also specifies which version of the application is available for a given session. All applications and libraries are made available through spack. A User has to load the appropriate package from the available packages.

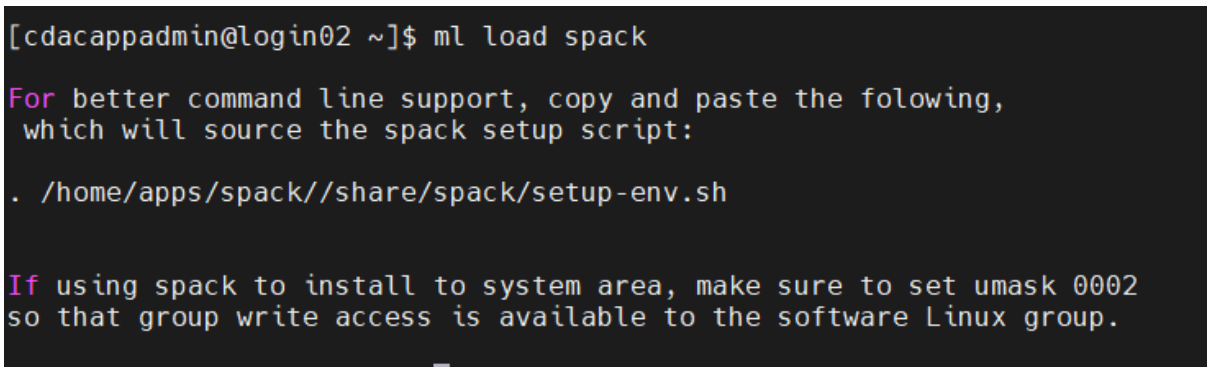
Introduction

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.

On your login node command prompt execute below commands:

```
$ module load spack
```

It will load SPACK module and set up environment for SPACK.



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 Kamrupa. 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.

```
[cdacappadmin@login02 ~]$ spack find
-- linux-almalinux8-cascadelake / gcc@12.2.0 --------------------
abseil-cpp@20240116.2        freetype@2.13.2              jasper@4.2.4
adios2@2.10.1                fribidi@1.0.12               json-glib@1.6.6
autoconf@2.69                g2c@1.9.0                    jsoncpp@1.9.5
autoconf-archive@2023.02.20  gawk@4.2.1                   kbproto@1.0.7
automake@1.16.1              gcc-runtime@12.2.0           krb5@1.21.2
bdftopcf@1.1                 gdbm@1.23                    krb5@1.21.2
```

spack load application name

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

```
[cdacappadmin@login02 ~]$ spack load gromacs
==> Error: gromacs matches multiple packages.
  Matching packages:
    fafxw2x gromacs@2024.2%gcc@12.2.0 arch=linux-almalinux8-cascadelake
    5p3fjl3 gromacs@2024.2%gcc@12.2.0 arch=linux-almalinux8-cascadelake
  Use a more specific spec (e.g., prepend '/' to the hash).
```

To know the Pre-Loaded Application/Compliers

```
[cdacappadmin@login02 ~]$ spack find --loaded
-- linux-almalinux8-skylake_avx512 / gcc@8.5.0 ------------------
cuda@11.8.0
==> 1 loaded package
[cdacappadmin@login02 ~]$
```

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.

```
[cdacappadmin@login02 ~]$ spack find --loaded
-- linux-almalinux8-skylake_avx512 / gcc@8.5.0 -----------------
cuda@11.8.0
==> 1 loaded package
[cdacappadmin@login02 ~]$ spack compilers
==> Available compilers
-- gcc almalinux8-x86_64 ---------------------------------------
gcc@8.5.0  gcc@13.2.0  gcc@12.2.0

-- intel almalinux8-x86_64 -------------------------------------
intel@2021.10.0

-- oneapi almalinux8-x86_64 ------------------------------------
oneapi@2024.2.0  oneapi@2023.2.0
[cdacappadmin@login02 ~]$
```

To check the compliers available in the system

```
[cdacappadmin@login02 ~]$ spack compiler list
==> Available compilers
-- gcc almalinux8-x86_64 ---------------------------------------
gcc@8.5.0  gcc@13.2.0  gcc@12.2.0

-- intel almalinux8-x86_64 -------------------------------------
intel@2021.10.0

-- oneapi almalinux8-x86_64 ------------------------------------
oneapi@2024.2.0  oneapi@2023.2.0
[cdacappadmin@login02 ~]$
```

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.

```
[appadmin@login01 ~]$ spack list
3dtk                                    perl-catalyst-runtime
3proxy                                  perl-catalyst-view-json
7zip                                    perl-cgi
abacus                                  perl-cgi-simple
abduco                                  perl-cgi-struct
abi-compliance-checker                  perl-chart-gnuplot
abi-dumper                              perl-chi
abinit                                  perl-chi-driver-memcached
abseil-cpp                              perl-class-accessor
abyss                                   perl-class-accessor-grouped
accfft                                  perl-class-accessor-lvalue
acct                                    perl-class-c3
accumulo                                perl-class-c3-adopt-next
ace                                     perl-class-c3-componentised
acfl                                    perl-class-data-inheritable
ack                                     perl-class-inspector
acl                                     perl-class-load
acpica-tools                            perl-class-load-xs
```

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. Refer Packaging Guide — Spack 0.22.0 documentation for detailed understanding of the Spack packaging.

Example Creating Own Package:

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

```
# 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 Banglore. """
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>)
```

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 the 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)	**8.5.0**, 9.3.0, 12.2.0, 13.2.0
icc/icpc/ifort	Intel Compilers (C/C++/Fortran)	2021.5.0, 2021.11.0, 2021.10.0 oneapi@2024.0.0, oneapi@2023.2.0, oneapi@2022.0.0
mpicc/mpicxx/mpif90	Intel-MPIwith GNU compilers (C/C++/Fortran)	2021.11.0
mpiicc/mpiicpc/mpiifort	Intel-MPIwithIntel compilers (C/C++/Fortran)	2021.11.0
nvcc	CUDA C Compiler	9.1.85, 11.8.0, 12.3.0

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:
spack load gcc@13.2.0 /irq6zpy
spack load intel-oneapi-compilers@2024
compilation: icc -O3 -xHost <<prog_name.c>>
Execution: ./a.out
```

C + OpenMP Program:

```
Setting up of environment:
spack load gcc@13.2.0 /irq6zpy
spack load intel-oneapi-compilers@2024
Compilation: icc -O3 -xHost -qopenmp <<prog_name.c>>
Execution: ./a.out
```

C + MPI Program:

```
Setting up of environment:
spack load gcc@13.2.0 /irq6zpy
spack load intel-oneapi-compilers@2024
Compilation: mpiicc -O3 -xHost <<prog_name.c>>
Execution: mpirun -n <<num_procs>> ./a.out
```

C + MKL Program:

```
Setting up of environment:
spack load gcc@13.2.0 /irq6zpy
spack load intel-oneapi-compilers@2024
Compilation: icc -O3 -xHost -mkl <<prog_name.c>>
Execution: ./a.out
```

CUDA Program:

```
Setting up of environment:
spack load gcc@12.2.0
spack load cuda@11

Example (1)
Compilation: nvcc -arch=sm_80<<prog_name.cu>>
Execution: ./a.out
Note: The optimization switch -arch=sm_80 is intended for NVIDIA A100 GPUs
and is valid for CUDA 11 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_80 /home/apps/Docs/samples/mm_blas.cu -lcublas
Execution: ./a.out
```

CUDA + OpenMP Program:

```
Setting up of environment:
spack load gcc@12.2.0
spack load cuda@11

Example (1)
Compilation: nvcc -arch=sm_80 -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
/home/apps/spack/opt/spack/linux-almalinux8-skylake_avx512/gcc-8.5.0/cuda-
11.8.0-6tjefbpfvo3ysl2dtqudbhsdgifpbgdh/include  -L
/home/apps/spack/opt/spack/linux-almalinux8-skylake_avx512/gcc-8.5.0/cuda-
11.8.0-6tjefbpfvo3ysl2dtqudbhsdgifpbgdh lib64 -lcublas
Execution: ./a.out
```

OpenACC Program:

```
Setting up of environment:
spack load nvhpc/ybz5nou

Compilation for GPU: pgcc -acc -fast -Minfo=all -
ta=tesla:cc80,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.
```

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 To

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 packgages.

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 commands 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 displays 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 watchpoints 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 a function call (only function or instruction compiled with debugging flags).

next:

This command also steps through the instructions of the program. Unlike the 'step' command, if the current source code line calls a subroutine, this command does not enter the subroutine, but instead steps over the call, in 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 15 – 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 16 – 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 operation. 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 the final random value is generated using a basic rand() modulo 10 operation.

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.

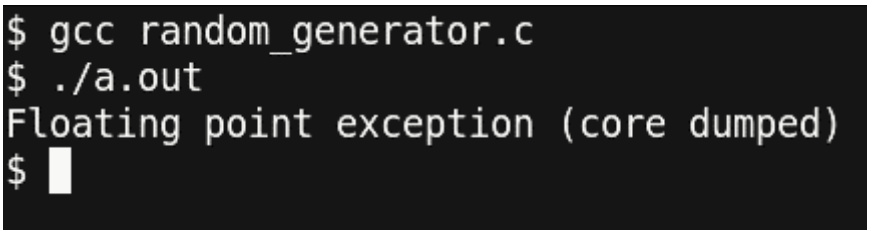


Figure 17- Output at a debugging stage

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 18 – 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 19 – 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 20 – 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 21 – 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 22 - 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; i < 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 23 – 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 the ‘print’ command we checked the values of ‘i’ and ‘j’. t 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 24 – 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 ‘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 25 – Debugging continued

You can see that we reduced the typing of the ‘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 26 – 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
0x0000000004011f5 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
0x0000000004011f5 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
0x0000000004011f5 in rand_fract () at random_generator.c:12
12                      for (short j = 1; i < N; ++j) {
(gdb)
```

Figure 27 – Setting a watchpoint

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 are called 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 28 – 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 has 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 29 – 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’ data type 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
164881519693493690771284741107526936387246517896865293689912664267996832785484381802480372560208997 7
$ ./a.out
Segmentation fault (core dumped)
$ ./a.out
569781955537763960836830241858826994391864733049244939153250232854585683358673709312240795711226896 3
$ ./a.out
615093049447589086305031871912273458286430976519379904084395812368188823030803931823443802406834874 7
$ ./a.out
Segmentation fault (core dumped)
$
```

Figure 30 – 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 31 – 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.
0x000000000040126c in factorial (x=4294792703) at random_generator.c:24
24                      return (x * factorial(x - 1));
(gdb)
```

Figure 32 – 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=0x7fffffffd0d8) 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 33 – Debugging continued (Will it ever end?)

Thus, we have set a breakpoint before the call of the function 'factorial' and run 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=0x7fffffffd0d8) 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.
0x000000000040126c in factorial (x=4294792699) at random_generator.c:24
24                      return (x * factorial(x - 1));
(gdb)
```

Figure 34 – 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=0x7fffffffd0d8) at random_generator.c:38
38              long long random_fact = factorial(f1);
(gdb) print f1
$2 = -8
(gdb)
```

Figure 35 – 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 36 – 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 functions 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 37 - 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 rerun and check our final program.

```
$ gcc random_generator.c
$ ./a.out
19471559044443562608278678958290139405601275743923843620615447570423185422006598995277439285952116 45
$ ./a.out
09299897455461678561009615129390185737602235048335425348860912942437328541267290962619417608015378 20
$ ./a.out
02442025927583905369914440384653965830535160224102285621881346655240493931055665005770058284870596 53
$ ./a.out
28727182932285670545393680969690664373798936715760291779097957013463932957649315367734833630351819 11
$ ./a.out
91287660615389560277595980747978327150874514377041221909658980833614136907231502145435177396365182 90
$ ./a.out
47005807923124125516733944531476306087909314926490273789232590252870772903316185104702628199316524 79
$ ./a.out
35979776328703654790231307059184469090834702633543759919836756312522527100583848415308484089632086 45
$ ./a.out
08645104190562912823688450791390952107926971917642093048030371586726511320524488687903019068128890 64
$ ./a.out
49729166095384459005299581582408490306127765102222753804974414253283808774506749236518905446082402 90
$ ./a.out
95286086428661777539838421822850471209841900007850956910192389646766662055067764070871803253117903 89
$
```

Figure 38 – Resolved

Thus, we are getting the correct results as expected.

Conclusion

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 problems causing bugs in large programs.

Overall Coding Modifications Done

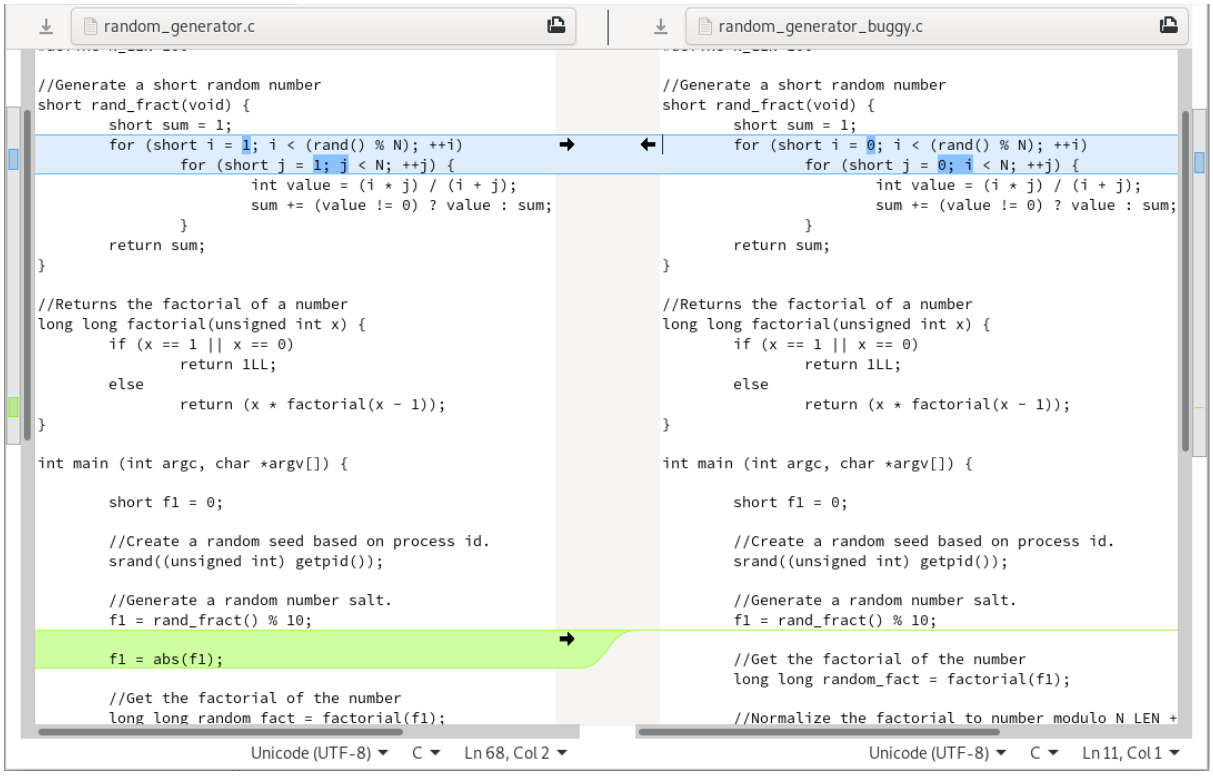


Figure 39 – What all we did to get things right !

Machine Learning (ML) / Deep Learning (DL) Application Development

Most of the popular python-based ML/DL libraries are installed on the PARAM Rudra system. Users while developing and testing their applications, can use conda based python installation.

For the conda environment 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 environments in the current bash shell and continuing with application development.

Once logged into PARAM Rudra HPC Cluster, check which all libraries are available, loaded in the 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
```

Defaults libraries and framework specific conda environment has been made available for user to start with application development which is installed with most of the popular python packages as shown below

Loading the Conda Base Module and Activating the Environments

In order to use base conda environment we first, access and load the miniconda module, which provides access to the base environment which is installed with default packages:

```
$ module load mldl/Miniconda
```

To see the list of other packages installed, use the command given below,

```
$ conda list
```

We provide multiple conda environments that include basic machine learning packages, as well as common image processing and natural language processing packages, for your machine learning projects.

The following table shows currently available conda environments with their version (all include GPU support):

	Frameworks	Environment	Version
DL Framework	Tensorflow	Tensorflow	2.15.0
	Tensorflow-gpu	Tensorflow-gpu	2.15.0
	Pytorch	Pytorch	2.2.0
	Pytorch-gpu	Pytorch-gpu	2.2.1
	Theano	Theano	1.0.5
	Theano-gpu	Theano-gpu	1.0.5
	Caffe	Caffe	1.0
	Caffe-gpu	Caffe-gpu	1.0
Distributed DL Framework	Horovod	Tensorflow	0.28.1
		Pytorch	0.28.1
Data science Framework	Rapids	Rapids	21.06

To activate any one of the environments we can load it on PARAM Rudra, load module "ENV_NAME" as shown below:

```
$ module load <ENV_NAME>
```

Once the "ENV_NAME" module is loaded, end-users can use all libraries inside their python program. Users can load those libraries using the "module load" command and use them for their applications.

Example: To activate Pytorch environment we can load it on PARAM Rudra, using module load Pytorch as shown below:

```
$ module load mldl/Pytorch
```

This will activate Pytorch environment in which users can use pytorch library and its related functionalities

Useful Conda Commands

After loading the module, you will have access to conda commands, including:

```
$ conda info --env
```

Shows available environments.

```
$ conda list -n env_name
```

Shows installed packages within an environment.

```
(env_name)$ conda deactivate
```

Deactivates an environment after loading.

For more detailed documentation, see the Conda website.

Managing and Installing Python Packages in Conda Environments

You have two options to install your own Python packages in our machine learning environment:

- Use the pip tool to install them directly
- Build your own conda environment

Consider the benefits and disadvantages of each method, before choosing which works best for you.

NOTE: Use Conda primarily for environment management, especially in scientific computing and data science projects where non-Python dependencies are common.

Use pip for installing Python packages from PyPI when you don't need the advanced environment management features provided by Conda.

Building Your Own Conda Environment

Building your own conda environment gives you the control to manage and install your own packages, and they will be less likely to have version errors than the pip-installed packages.

The easiest way to create your own environment is to clone an existing conda environment into your own directory, then modify it.

Creating an environment can take up a significant portion of your disk quota, depending on the packages installed. To ensure that you can use your conda environment properly, please familiarize yourself with all the basic conda commands.

Conda based installation provides the latest version of DL framework, however users can install their own choice of DL framework or library version locally by following below steps.

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

Step 2. Activate conda environment.

```
$ module load mldl/Miniconda
```

Step 3. Create the local environment myenv (myenv is the environment name, you can give any name of your choice).

```
$ conda create --name myenv
```

Step 4. Activate a newly created environment.

```
$ conda activate myenv
```

Step 5. Install your own DL framework / python library. <package-name> will get replaced by desired package which user wants to install

$ conda install <package-name>

Example: In order to install numpy we can use below command.

```
$ conda install numpy
```

Now you can use the newly installed package in your python program.

Submitting job using sbatch script for DL Application

You can activate your machine learning environment, run your program, and deactivate the environment in a SLURM sbatch script. For example:

```
#!/bin/bash -x
#SBATCH -N 1
#SBATCH --ntasks-per-node=<np>
```

```
#SBATCH -p cpu
#SBATCH -J <job_name>
#SBATCH -t 05:00:00
#SBATCH -o %j.out              # name of stdout output file(--output)
#SBATCH -e %j.err              # name of stderr error file(--error)
cd $SLURM_WORKDIR
module purge
module load mldl/Miniconda          # load the module and environment
conda activate <env_name>      # load working environment
python <script>.py             # run python script
conda deactivate         # deactivate environment
# end of script
```

How to launch a Jupyter notebook?

You can access the Jupyter notebook from your local system, while it is actually running under the conda virtual environment, setup on a remote server. It can be accessed through the browser of your local system using ssh tunneling technique.

NOTE: To launch the jupyter notebook from gpu, first login to gpu node using the below command in the login node.

```
$ salloc --nodes=1 --time=1:00:00 --partition=gpu
```

To check which gpu node is assigned, use the below command.

```
$ squeue --me
```

Now ssh to the node assigned to you. For example, in the screenshot below, you can see that gpu007 was assigned to the user.

```
$ ssh gpu007
```

Now to launch the notebook from the gpu node, follow the below steps.

1. Activate the Conda environment.

To submit the job, use the below command.

```
$ module load mldl/Miniconda
```

2. Start the jupyter notebook by below command.

```
(base)$ jupyter notebook --ip=0.0.0.0 --port=<PORT_NO> --allow-root --no-
browser
```

For example,

```
(base)$ jupyter notebook --ip=0.0.0.0 --port=8888 --allow-root --no-browser
```

Note: Token number displayed on the screen would later be used for login to jupyter notebook through your local web browser.

3. From another terminal, on your mobaxterm, create ssh tunneling between your local machine and remote system by executing below command

```
$ ssh -t -t username@<IP_ADDR> -L <PORT_NO>:localhost:<PORT_NO> ssh gpu<NO>
-L <PORT_NO>:localhost:<PORT_NO>
```

For example,

```
$ ssh -p 4422 -t -t appsupport@<IP_ADDR> -L 8888:localhost:8888 ssh gpu007
-L 8888:localhost:8888
```

Note: Use the port number and gpu node that is assigned by slurm.

4. Type the below address in your local browser to access Jupyter notebook.

```
https://localhost:<PORT_NO>
```

For example,

```
https://localhost:8888
```

Note: Enter token number for login

5. The Jupyter notebook can now be opened after entering the valid token

Some Important Facts

About File Size

The global space is served by a number of storage arrays. Each of the storage array contains a portion of the space. 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 a 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 subdirectories 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 following command:

lfs getstripe <path to the direcory>

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 on 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 jobs. It is best to run the job on 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 an application may cause linking or execution issues.
4. Choosing appropriate compiler switches/flags/options (e.g. –O3) may increase the performance of the 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 Rudra 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 Rudra

The purpose of this section is to expose the users to different application packages which have been installed on PARAM Rudra System. 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 many others. More information about LAMMPS may please be found at https://lammps.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
spack load intel-oneapi-compilers /jtvke3n
spack load intel-oneapi-mpi/2db2e7t
spack load gcc@13.2.0/3wdooxp
source /home/apps/SPACK/spack/opt/spack/linux-almalinux8-cascadelake/gcc-
13.2.0/intel-oneapi-mkl-2024.0.0-
yq4keqsjr44rf5ffroiiim2iklxg4let/setvars.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
```

```
#Enter your working directory or use SLURM_SUBMIT_DIR
cd /home/manjunath/NEW_LAMMPS/lammps-7Aug19/bench

export OMP_NUM_THREADS=1
time mpiexec.hydra -n $SLURM_NTASKS -genv OMP_NUM_THREADS 1 <path of lammps
executable> -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 centres 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

source /home/apps/spack/share/spack/setup-env.sh

spack load intel-oneapi-compilers /glrrtsv
spack load gromacs@2024.2 /5p3fjl3

#Enter your working directory or use SLURM_SUBMIT_DIR
cd /home/shweta/water-cut1.0_GMX50_bare/3072

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 RUDRA' at Aruna Asaf Ali Marg, near Vasant Kunj, Vasant Kunj, New Delhi, Delhi 110067, 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:

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 the 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 the National Supercomputing Mission.

Getting Help – PARAM Rudra 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 Rudra 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 issues that you are experiencing.

Steps to Create a New Ticket

1. Place the URL (https://paramrudra.bose.res.in/support) in your browser.
2. On the right-top corner of the page click **Sign In**. Refer to Fig: 36 for the same.

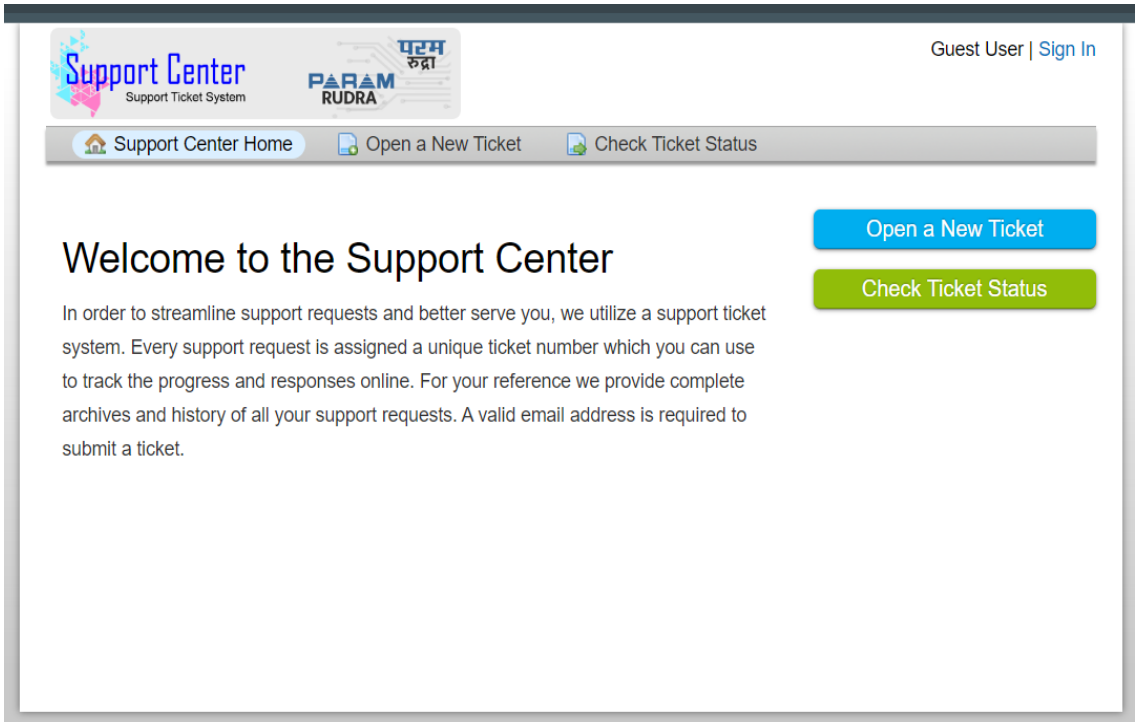


Figure 40 – Snapshot of Ticketing System

3. Sign in by using the Username and Password that you use for logging to the Cluster. Refer to Fig37 for the same.

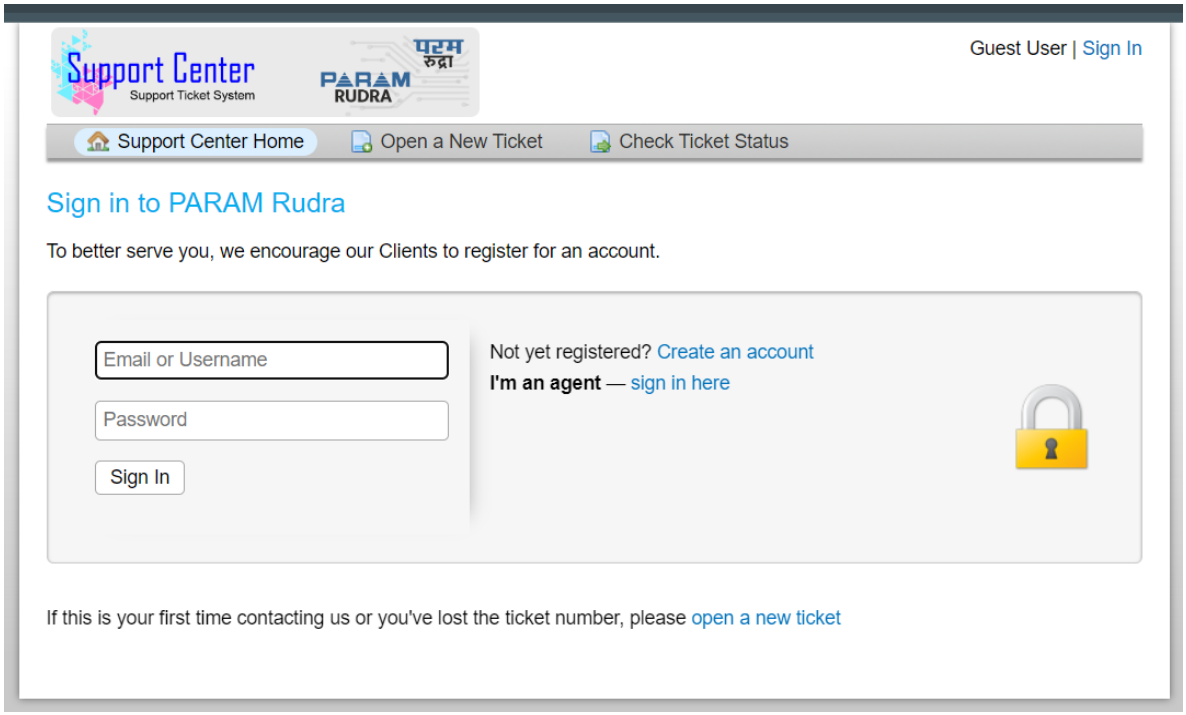


Figure 41- Snapshot of Ticketing System

4. Select a **Help Topic** from the Dropdown and then Click on **Create Ticket**. Refer to Fig:38 for the same

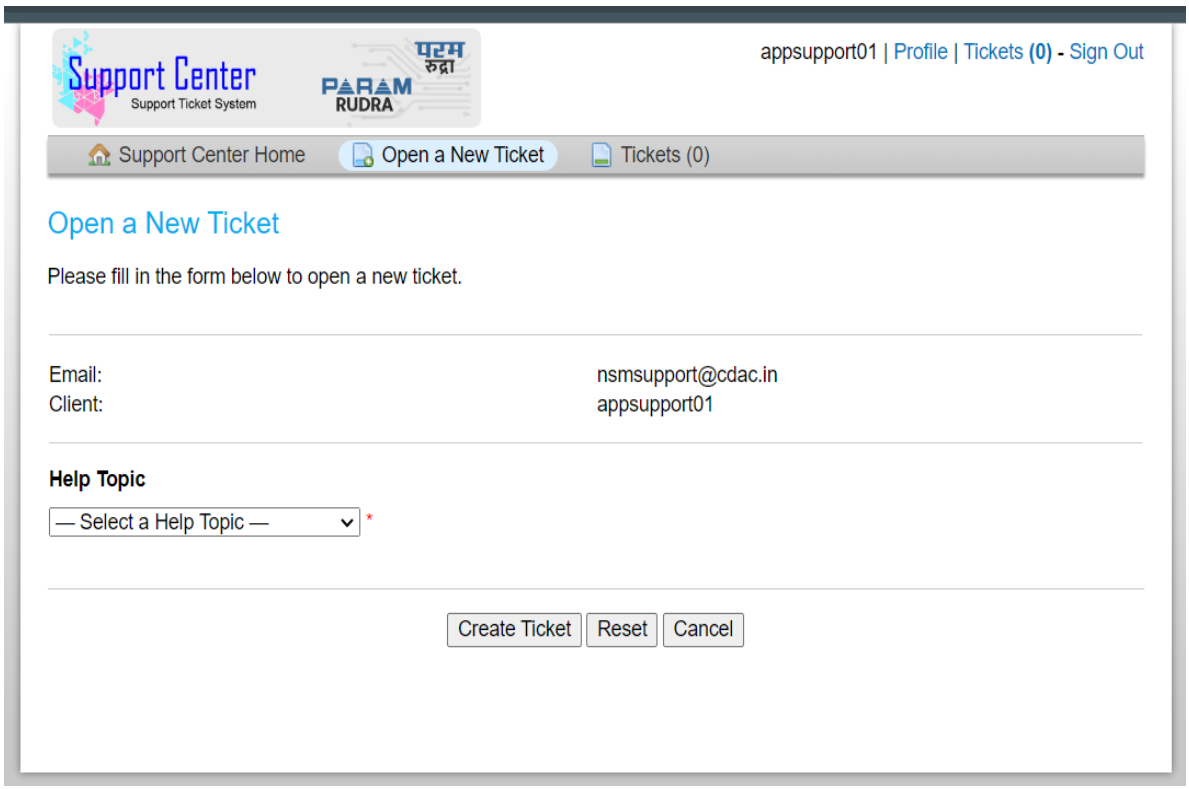


Figure 42 - Snapshot of Ticketing System

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

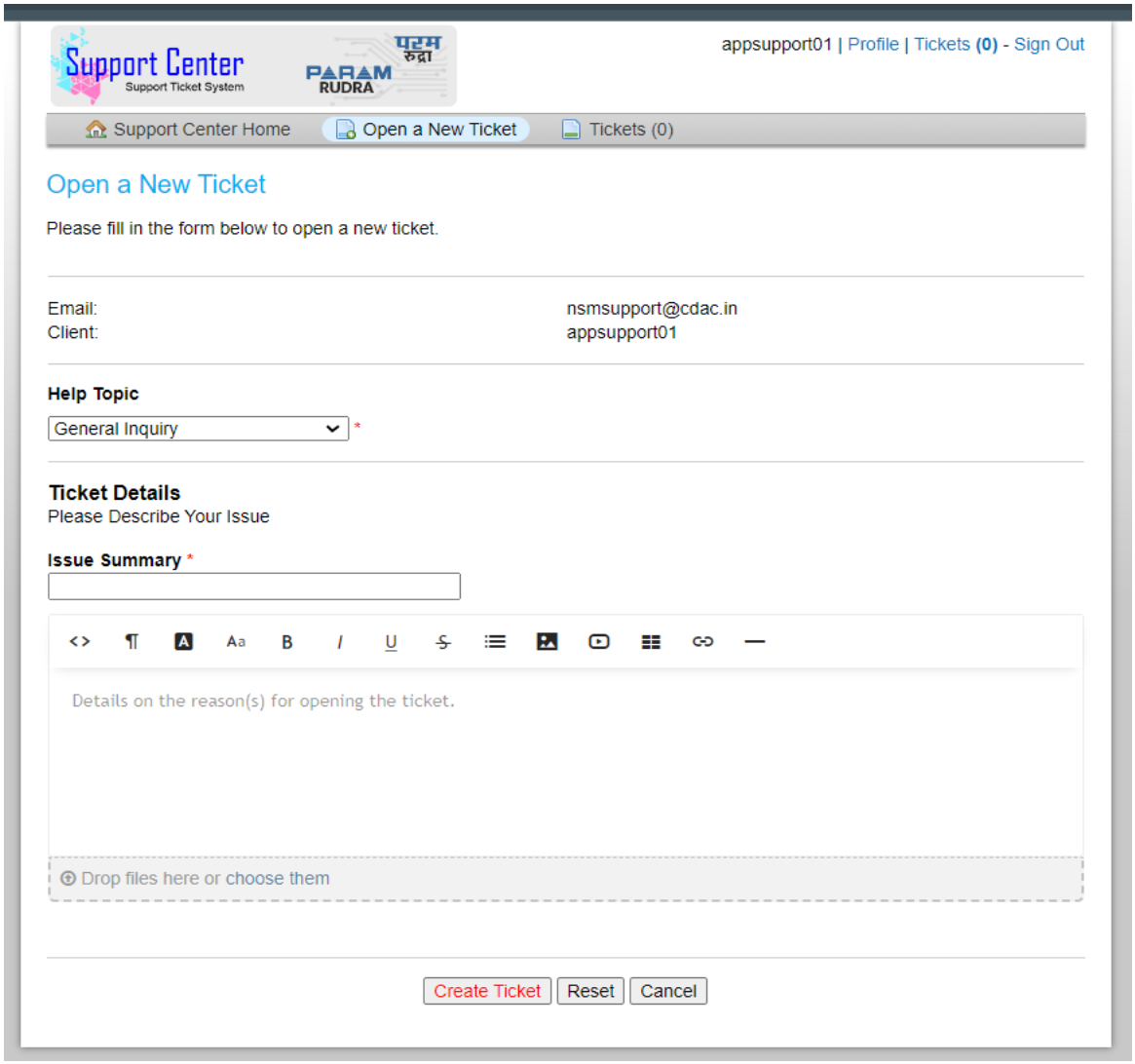


Figure 43 - Snapshot of Ticketing System

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 Rudra Support you can reach us via e-mail at rudrasupport@iuac.res.in

User Creation Process

To get access to this HPC Facility, proceed with registration on the Portal through the link provided below:

Link: https://services.nsmindia.in/userportal/account

Once registered, you will receive an email outlining the next steps to be followed for your User Creation Request.

User Creation Portal streamlines the user data collection process, enabling multiple users to submit user creation requests simultaneously. Physical form maintenance is eliminated; users need to provide accurate official details and an email address for procedural notifications. Users can track their account creation status, remaining steps, and identify necessary actions. Administrative or Higher authorities can access all user details through secure login into the portal.

Process/Steps

Users initiate registration on the portal by entering their email address, city, and institute name. An email will be sent to verify the provided email address, upon verification registration form link is sent for completing the user account request. Users have the option to preview and edit the form before the final submission. Upon submission, a link for Document Upload is provided, where documents like ID proof, User Creation Form and other needed documents are uploaded. Once documents are uploaded, modifications are not possible as the documents will undergo verification processes.

User details and documents undergo verification by the user's Institute HOD/PI. Upon approval, a verification email is sent to the coordinator. The coordinator selects the appropriate cluster for the user based on document verification and requirements. Final approval is granted by higher authority, resulting in acceptance of the user request.

If you have any queries, refer to the User Creation Manual and Flowcharts accessible in the Help section within the User Creation Portal. Furthermore, common questions are addressed in the FAQ section located beside the Help section. If you have any additional inquiries or require assistance, feel free to reach out to us at **nsmsupport@cdac.in**.

Note: Kindly use your official email address for registration to avoid the possibility of your request being declined.

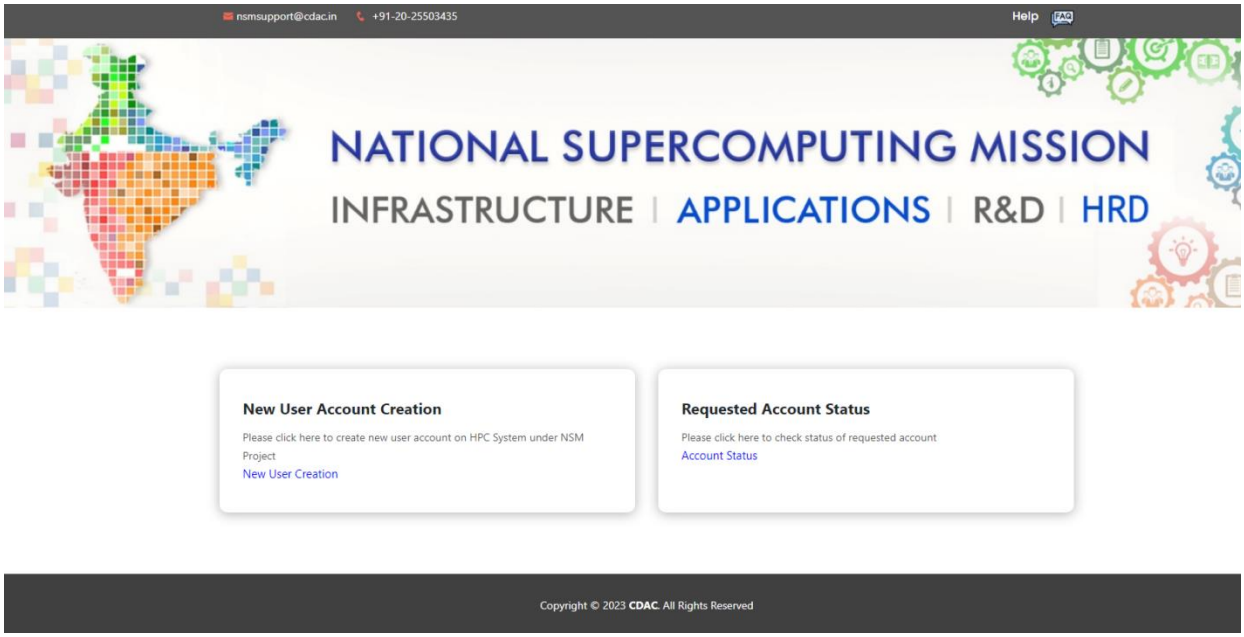


Figure 43 - Snapshot of Ticketing System

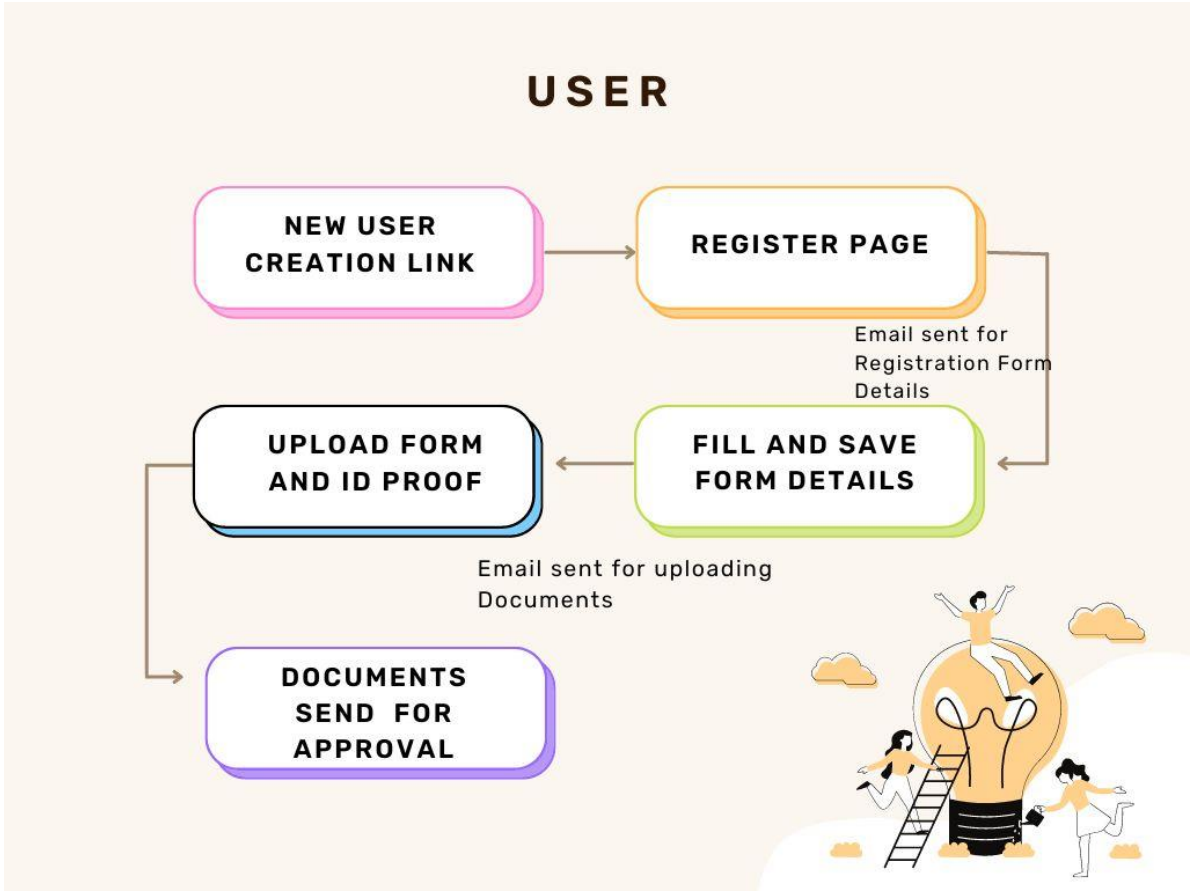


Figure 44 - User Flow

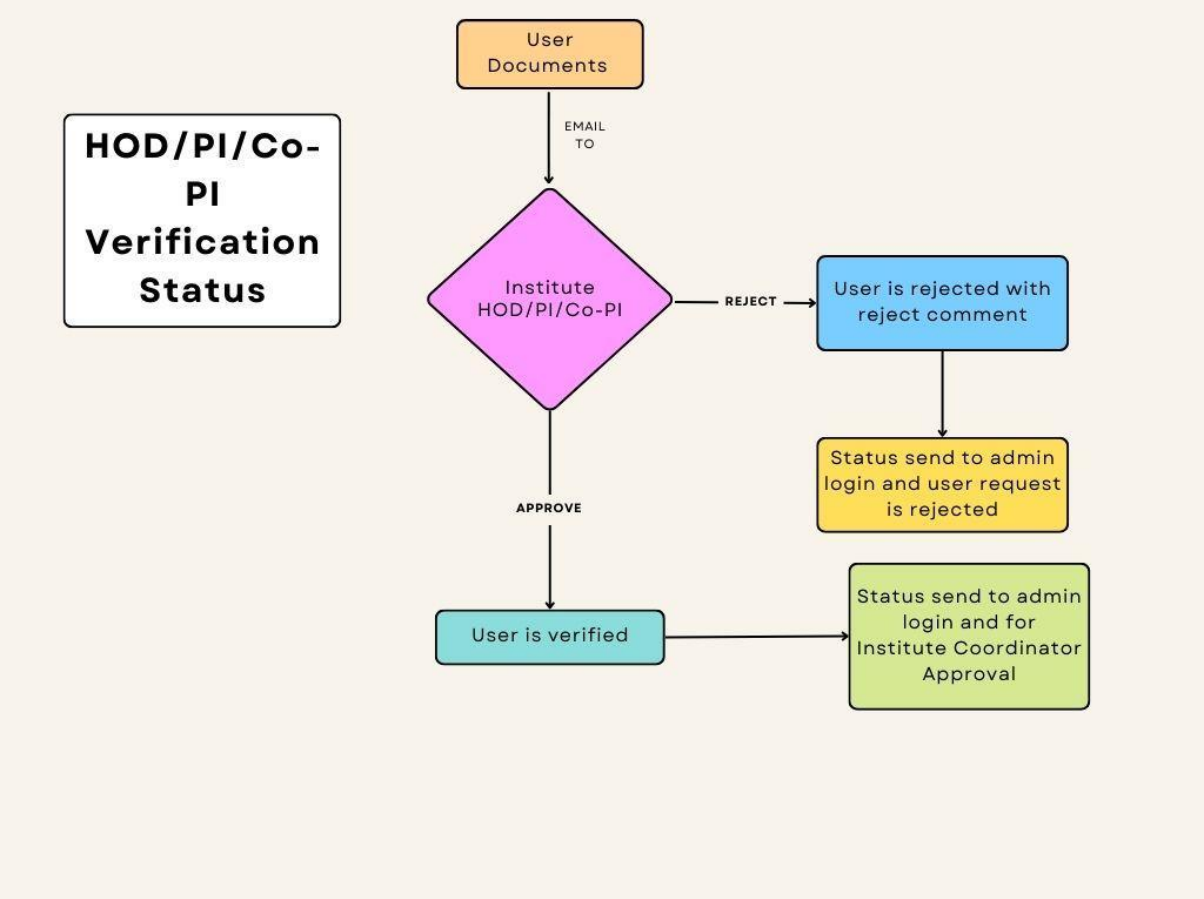


Figure 45 - HOD/PI/Co-PI verification Flow

Closing Your Account on PARAM Rudra

When once you have completed your research work and you no longer need to use PARAM Rudra, you may please close your account on PARAM Rudra. Please raise a ticket by following the URL https://paramrudra.iuac.res.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.

References

1. LAMMPS (Molecular Dynamics Simulations) https://lammps.sandia.gov/
2. https://www.openacc.org/
3. https://www.openmp.org/
4. BLAST (Basic Local Alignment Search Tool) https://blast.ncbi.nlm.nih.gov/Blast.cgi
5. VASP (Vienna Ab initio Simulation Package) https://www.vasp.at/
6. Gaussian (Computational Chemistry Software) https://gaussian.com/
7. https://computing.llnl.gov/tutorials/mpi/
8. CUDA (Parallel Computing Platform and API) https://developer.nvidia.com/cuda-zone
9. https://www.mmm.ucar.edu/weather-research-and-forecasting-model
10. GROMACS (Molecular Dynamics Simulations) http://www.gromacs.org/
11. OpenFOAM (Computational Fluid Dynamics) https://www.openfoam.com/
12. SLURM (Simple Linux Utility for Resource Management) https://slurm.schedmd.com/
13. https://www.tutorialspoint.com/gnu_debugger/what_is_gdb.htm
14. https://nsmindia.in/
15. https://en.wikipedia.org/wiki/Deep_learning
16. https://docs.conda.io/en/latest/miniconda.html
17. https://www.tensorflow.org/
18. https://github.com/PaddlePaddle/Paddle
19. Keras, https://keras.io/
20. Pytorch, https://pytorch.org
21. https://mxnet.apache.org
22. https://software.intel.com/en-us/distribution-for-python
23. https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide
24. NAMD (Molecular Dynamics Simulations) https://www.ks.uiuc.edu/Research/namd/
25. ANSYS (Engineering Simulation Software) https://www.ansys.com/
26. MPI (Message Passing Interface) https://www.mpi-forum.org/
27. AMBER (Assisted Model Building with Energy Refinement) https://ambermd.org/
28. CHARMM (Chemistry at HARvard Macromolecular Mechanics) https://www.charmm.org/