PARAM Sanganak

User’s Manual

Ver. 1.1.1

Last updated: March 09, 2021

www.cdac.in

Copyright Notice

Trademarks

Intended Audience

This document is meant for PARAM Sanganak 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 sanganaksupport@iitk.ac.in .

Give us your feedback

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

DISCLAIMER

Table of Contents

List of Figures

Introduction

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

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

It consists of 2 Master nodes, 8 Login nodes, 10 Service/Management nodes and 312 (CPU+GPU) nodes with total peak computing capacity of 1.66 (CPU+GPU) PFLOPS performance.

System Architecture and Configuration

System Hardware Specifications

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

- Total number of nodes: 332 (20 + 312)
    - Service nodes: 20**(Master+ Login+ Service+ Management Nodes)
    - CPU only nodes: 150
    - GPU ready nodes: 64
    - GPU nodes: 20
    - High Memory nodes:78

Master Nodes: 2

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

Master Nodes: 2	

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

Login Nodes: 8

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 and interactive job or submit batch jobs through the batch system (SLURM) to run your jobs on compute nodes. For ALL users PARAM Sanganak 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: 8	

2* Intel Xeon G-6248
Cores = 40, 2.5 GHz
Memory= 384 GB
HDD = 1 TBx8
	Total Cores = 320 cores
	
	Total Memory = 3072 GB

Service/Management Nodes: 10

Typically, the purpose of the service node is to provide Job Scheduling Services and other services to the cluster.

Service Nodes: 4	

2* Intel Xeon G-6248
Cores = 40, 2.5 GHz
Memory= 384 GB
HDD = 1 TBx5
	Total Cores = 160 cores
	
	Total Memory= 1536 GB

CPU Compute Nodes: 214

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

CPU only Compute Nodes: 150	

2* Intel Xeon Platinum 8268
Cores = 48, 2.9 GHz
Memory= 192 GB, DDR4 2933 MHz
SSD = 480 GB (local scratch) per node
	Total Cores = 7200 cores
	
	Total Memory=28800 GB

GPU ready Compute Nodes: 64	

2* Intel Xeon Platinum 8268
Cores = 48, 2.9 GHz
Memory= 192 GB, DDR4 2933 MHz
SSD = 480 GB (local scratch) per node
	Total Cores = 3072 cores
	
	Total Memory=12,288 GB

High Memory nodes: 78

Some compute nodes may feature a particular specification to be used for a particular job, or stage in your workflow.

These are High Memory nodes that provide users to run their memory intensive jobs.

CPU only Compute Nodes with High memory: 78	
2* Intel Xeon Platinum 8268	
Cores = 48, 2.9 GHz	Total Cores = 3744 cores
Memory= 768 GB, DDR4 2933 MHz	Total Memory=59904 GB
SSD = 480 GB (local scratch) per node	

GPU Compute Nodes: 20

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

GPU Compute Nodes: 20	
2* Intel Xeon G-6248	Total Cores = 800 cores
Cores = 40, 2.5 GHz	
Memory= 192 GB, DDR4 2933 MHz	Total Memory= 3840 GB
SSD = 480 GB (local scratch) per node	
2*NVidia V100 per node	
GPU Cores per node= 2*5120= 10240	
GPU Memory = 16 GB HBM2 per NVidia V100	

Storage

- Based on Lustre parallel file system
- Total useable capacity 2.2 PiB primary storage
- Throughput 50 GB/s

Operating System

- Operating system on PARAM Sanganak is Linux – CentOS 7.6

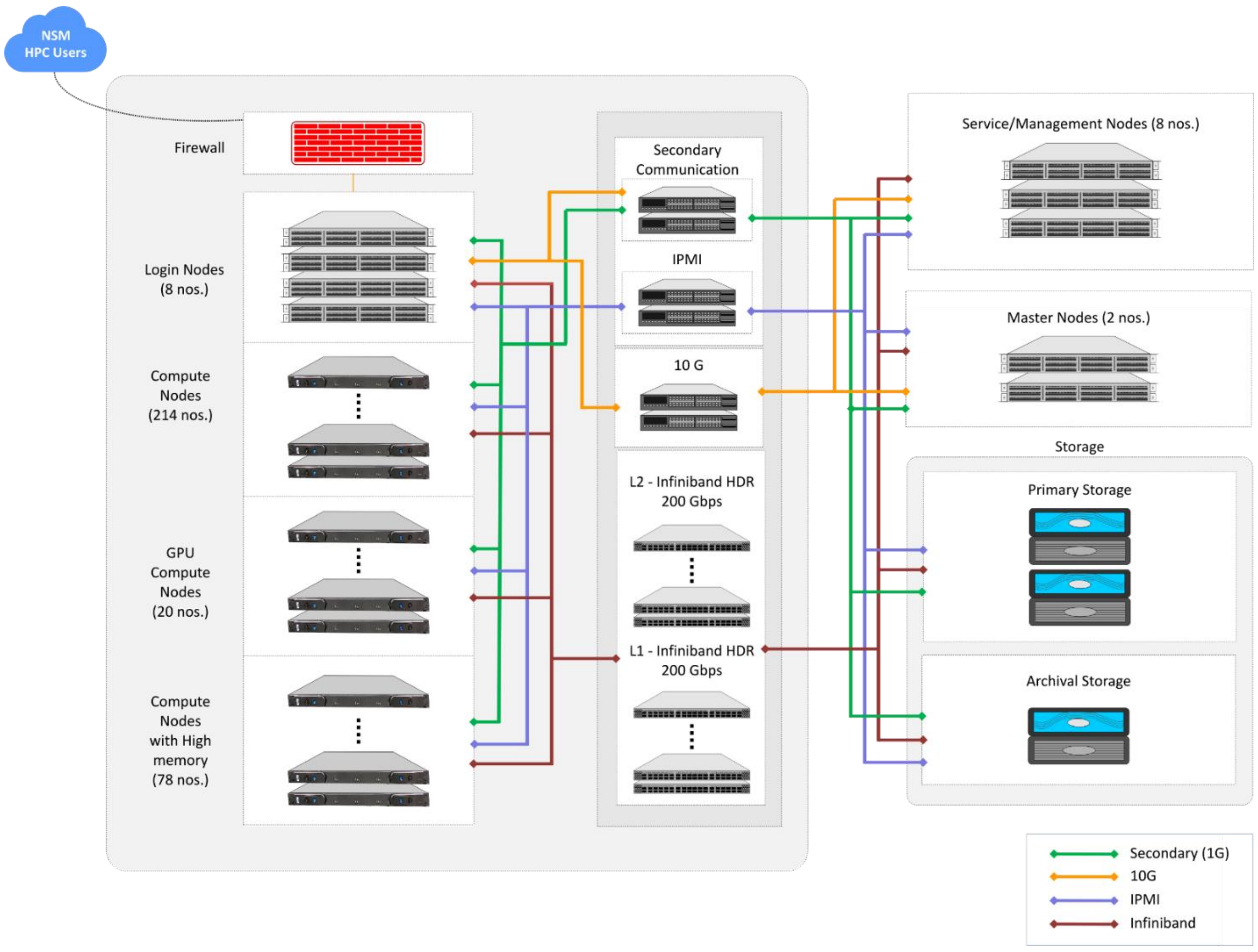


Figure 1 -PARAM Sanganak Architecture Diagram

Network infrastructure

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

a) Management functionalities i.e., to monitor, troubleshoot, start, stop various components of the cluster, etc. (Network/ portion of Network which implements this functionality is referred to as Management fabric).

b) Ensuring fast read/ write access to the storage (Network/ portion of Network which implements this functionality is referred to as storage fabric).

c) Ensuring fast I/O operations like connecting to other clusters, connecting the cluster to various users on the campus LAN, etc. (Network/ portion of Network which implements this functionality is referred to as I/O Fabric).

d) Ensuring High-Bandwidth, Low-latency communication amongst processors to for achieving high-scalability (Network/ portion of Network which implements this functionality is referred to as Message Passing Fabric)

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

Primary Interconnection Network

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

InfiniBand: 100 Gbps

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

Secondary Interconnection Network

Gigabit Ethernet: 1 Gbps

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

Software Stack

Software stack is an aggregation of software components that work in tandem to accomplish a given task. The task can be, to facilitate a user to execute his job/s or to facilitate a system administrator to manage a system efficiently. In effect, the software will have all the necessary components to accomplish a given task. There may be multiple components of different flavors to accomplish a given sub-task. The user/ administrator may mix and match these components depending on his choice. Typically, a user would be interested in preparing his executables, executing the same with his data sets and visualize the output generated by him. For accomplishing the same, the user would need to compile his codes, link the codes with communication libraries, Math Libraries, Numerical algorithm libraries, prepare the executables, run the same with desired data sets, monitor the progress of his jobs, gathering the results and visualizing the output.

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

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

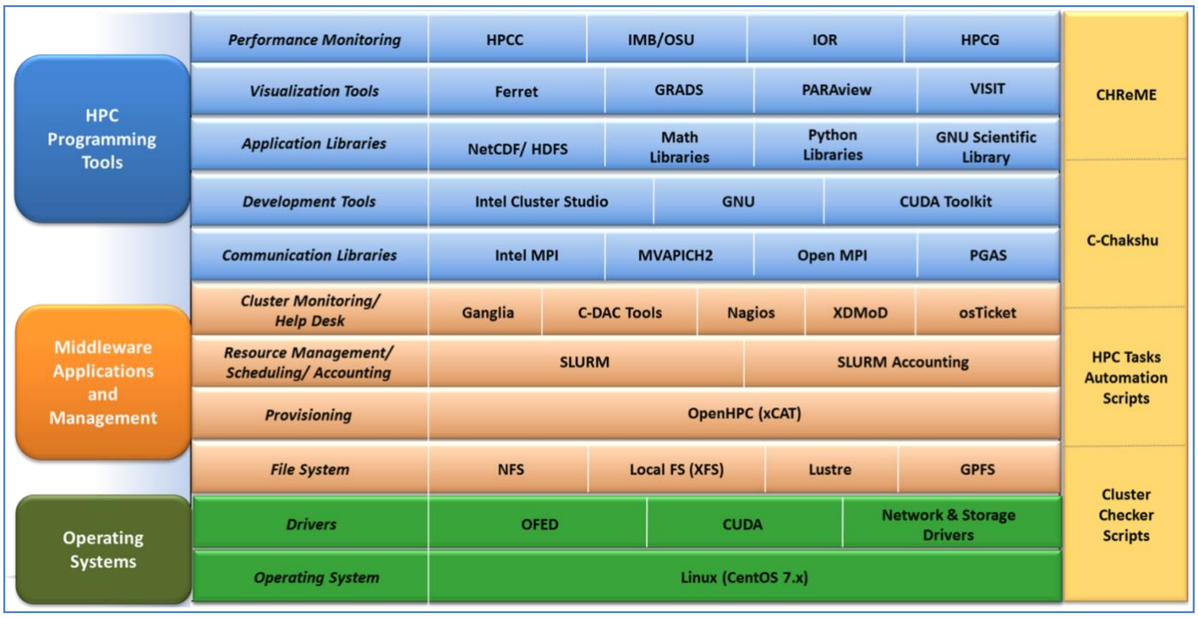


Figure 2 – Software Stack

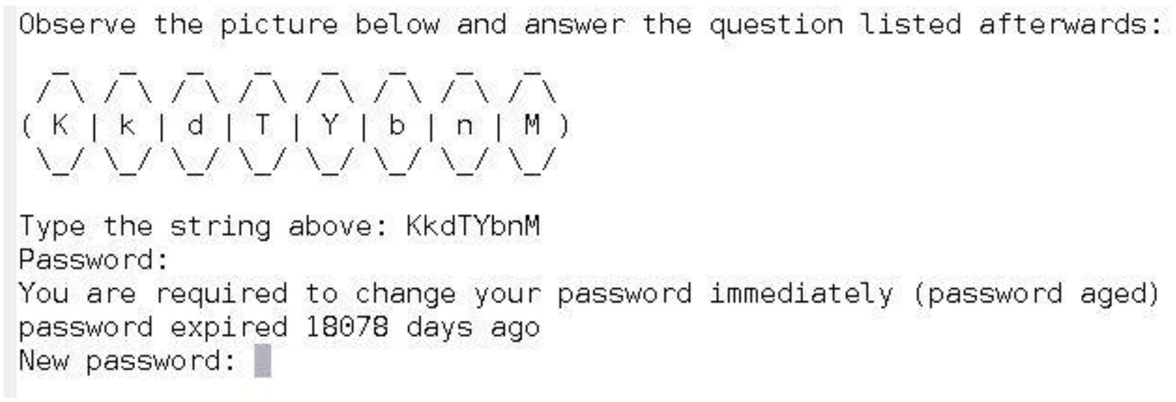
Functional Areas	Components
Base OS	CentOS 7.6
Architecture	X86_64
Provisioning	xCAT 2.14.6
Cluster Manager	OpenHPC (ohpc-xCAT 1.3.8)
Monitoring Tools	C-CHAKSHU, Nagios, Ganglia, XDMoD
Resource Manager	SLURM
I/O Services	Lustre Client
High Speed Interconnects	Mellanox InfiniBand
Compiler Families	GNU (gcc, g++, gfortran) Intel Compiler (icc, ifort, icpc)
MPI Families	MVAPICH, OpenMPI, MPICH

First Things First

First login

Whenever the newly created user on PARAM Sanganak tries to login with the user Id and password (temporary, system generated) provided over the Email through PARAM Sanganak support, he/she will next be prompted to create a “new password” of their choice which will change the temporary, system generated password. This will enable you to keep your account secure. It is recommended that you have a strong password which contains the combination of alphabets (lower case / upper case), numbers, and a few special characters that you can easily remember.

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



```
Observe the picture below and answer the question listed afterwards:
  _   _   _   _   _   _   _   _
 / \ / \ / \ / \ / \ / \ / \ / \
( K | k | d | T | Y | b | n | M )
 \_/ \_/ \_/ \_/ \_/ \_/ \_/ \_/

Type the string above: KkdTYbnM
Password:
You are required to change your password immediately (password aged)
password expired 18078 days ago
New password:
```

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

Forgot Password?

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

1. Open the PARAM Sanganak support site i.e. the ticketing tool by following the link https://paramsanganak.iitk.ac.in/support
2. Login with your registered email id, Complete name, Contact number.
3. There you can raise a ticket to get the password reset.

4. The system admin person will revert with an email for verification.
5. Once acknowledged, the password will be reset for the user and an email will be sent back intimating same.
6. Then the user can login with the temporary password and can set a new password of his/her choice.

System Access

Accessing the cluster

The cluster can be accessed through 4 general login nodes, which allows users to login.

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

Remote Access

Using SSH in Windows

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

(e.g. [username]@paramsanganak.iitk.ac.in)

Enter your password when prompted, and press Enter.

Using SSH in Mac or Linux

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

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

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

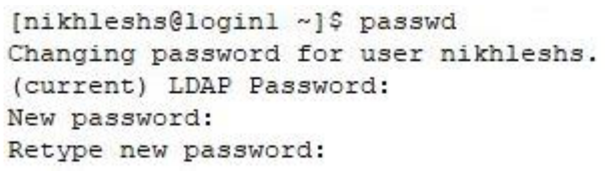
```
user1: ssh user1@paramsanganak.iitk.ac.in
```

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

Password

How to change the user password?

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



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

Transferring files between local machine and HPC cluster

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

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

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

```
/scratch/<username>/: ! This directory is user to store the user data
related to the project/research.
```

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

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

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

```
Scp –r <path to the local data directory>   <your username>@<IP of
paramSanganak>:<path to directory on HPC where to save the data>
```

Example:

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

```
Scp  –r  /dir/dir/file  saurabh@<cluster IP/Name>:/home/Saurabh
```

Example:

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

```
Scp  –r  /home/saurabh  saurabh@<local system IP/Name>:/dir/dir/file
```

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

To reiterate,

Copying Directory/File from local machine to PARAM Sanganak:

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

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

   user1@mylaptop:~$cd ~/MyData/

2. Under parent directory type ls <& press Enter key>, & notice Wrf-2.0 is there. user1@mylaptop: ~$ls Files TempFiles-0.5 Wrf-2.0
3. Begin copy by typing:

   user1@mylaptop: ~$ scp -r Wrf-2.0 (username)@paramsanganak.iitk.ac.in

   < you will be prompted for password; enter your password >

4. Now login to your account as: user1@mylaptop: ~$ ssh (your username) @ paramsanganak.iitk.ac.in < you will be prompted for password ; enter password > [user1@login ~]$
5. ls command, you should see Wrf-2.0 directory.
6. While copying from PARAM Sanganak to your local machine, follow the same steps.

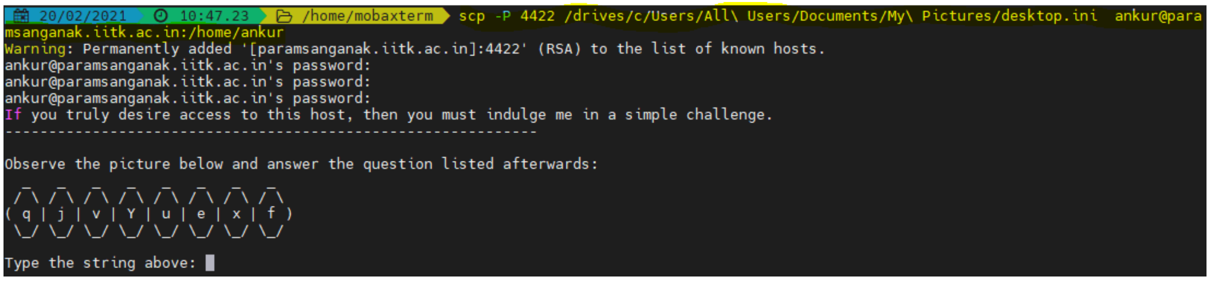
By interchanging source and destination in the scp command. Refer to the generic copying described earlier.

Tools

MobaXterm (Windows installable application):

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

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

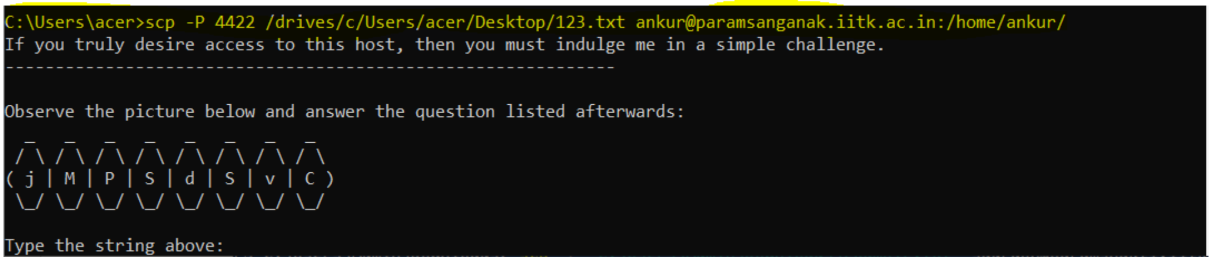


```
 20/02/2021   10:47.23   /home/mobaxterm  scp -P 4422 /drives/c/Users/All\ Users/Documents/My\ Pictures/desktop.ini  ankur@para
msanganak.iitk.ac.in:/home/ankur
Warning: Permanently added '[paramsanganak.iitk.ac.in]:4422' (RSA) to the list of known hosts.
ankur@paramsanganak.iitk.ac.in's password:
ankur@paramsanganak.iitk.ac.in's password:
ankur@paramsanganak.iitk.ac.in's password:
If you truly desire access to this host, then you must indulge me in a simple challenge.
--------------------------------------------------------------

Observe the picture below and answer the question listed afterwards:
  _   _   _   _   _   _   _   _
 / \ / \ / \ / \ / \ / \ / \ / \
( q | j | v | Y | u | e | x | f )
 \_/ \_/ \_/ \_/ \_/ \_/ \_/ \_/

Type the string above: █
```

Figure 3 - A snapshot of command using MobaXterm

Command Prompt (Windows native application):

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

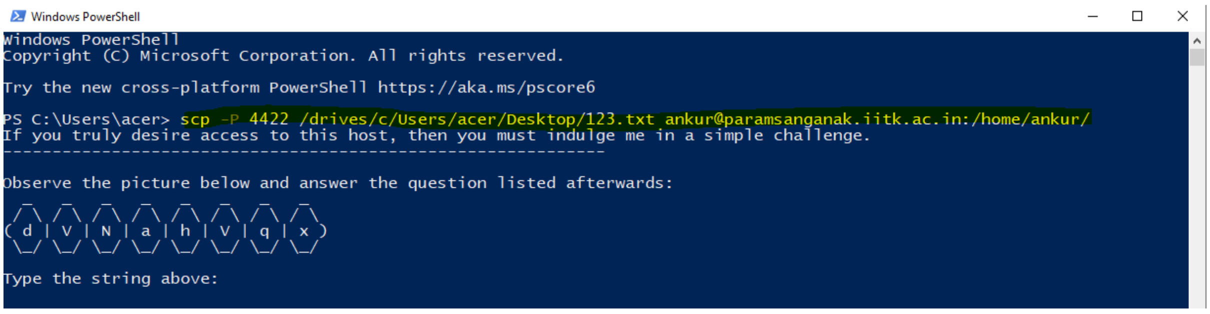


```
C:\Users\acer>scp -P 4422 /drives/c/Users/acer/Desktop/123.txt ankur@paramsanganak.iitk.ac.in:/home/ankur/
If you truly desire access to this host, then you must indulge me in a simple challenge.
--------------------------------------------------------------

Observe the picture below and answer the question listed afterwards:
  _   _   _   _   _   _   _   _
 / \ / \ / \ / \ / \ / \ / \ / \
( j | M | P | S | d | S | v | C )
 \_/ \_/ \_/ \_/ \_/ \_/ \_/ \_/

Type the string above:
```

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

PowerShell (Windows native application):

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



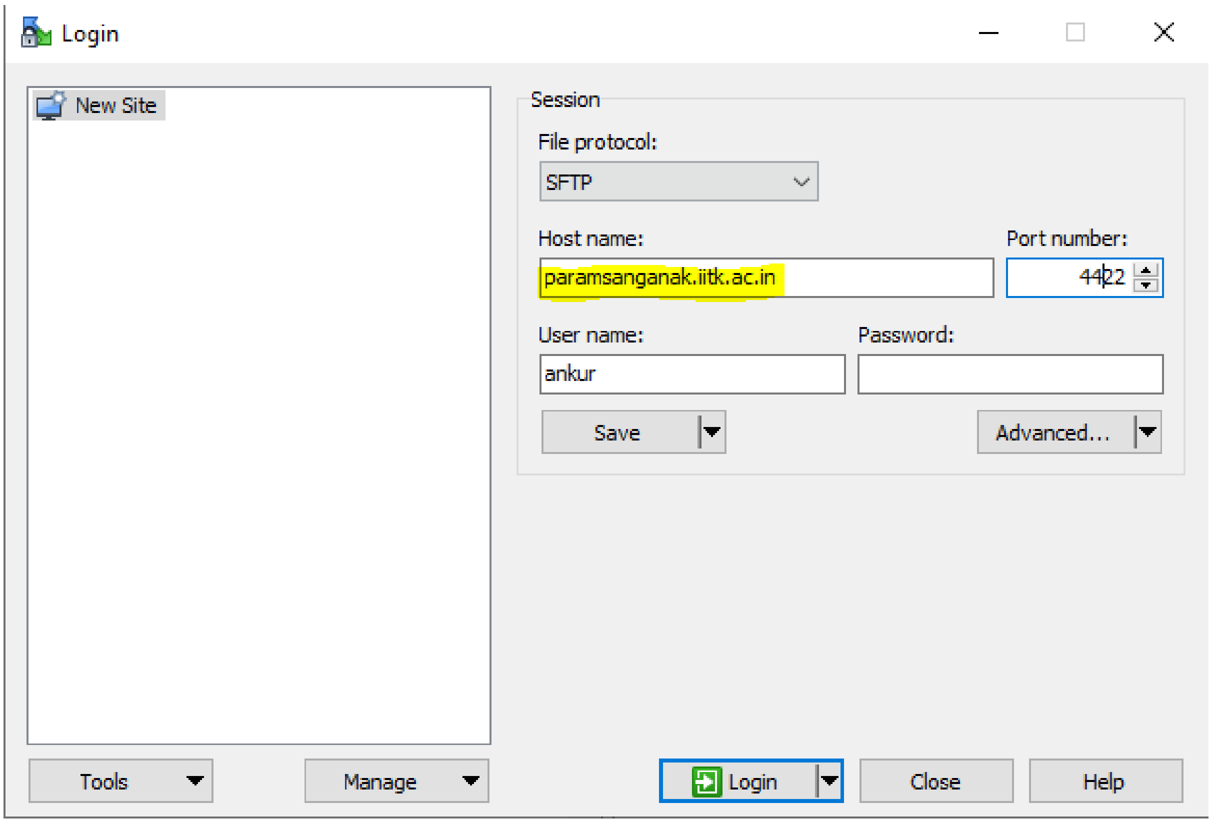
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\acer> scp -P 4422 /drives/c/Users/acer/Desktop/123.txt ankur@paramsanganak.iitk.ac.in:/home/ankur/
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:
  _   _   _   _   _   _   _   _
 / \ / \ / \ / \ / \ / \ / \ / \
( d | V | N | a | h | V | q | x )
 \_/ \_/ \_/ \_/ \_/ \_/ \_/ \_/

Type the string above:
```

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

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



Login

New Site

Session

File protocol: SFTP

Host name: paramsanganak.iitk.ac.in

Port number: 4422

User name: ankur

Password:

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

Running Interactive Jobs

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

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

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

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

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

Where xxxxx is the job id.

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

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

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

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

Managing Jobs through its Lifecycle

PARAM Sanganak extensively uses modules. The purpose of module is to provide the production environment for a given application, outside of the application itself. This also specifies which version of the application is available for a given session. All applications and libraries are made available through module files. A User has to load the appropriate module from the available modules.

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

module load compiler/intel/2018.2.199       # This will load the intel compilers into your
environment

module unload compiler/intel/2018.2.199   # This will remove all environment setting
related to intel-2018 compiler loaded previously
```

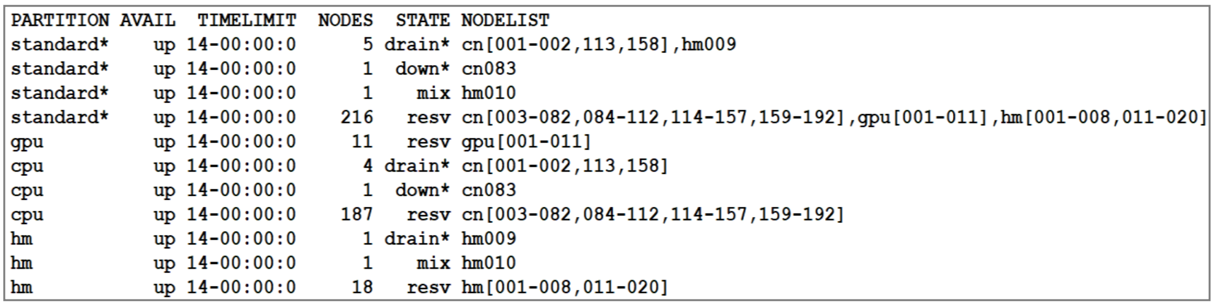
A simple Slurm job script

```
#!/bin/sh
#SBATCH -N 16                           // specifies number of nodes
#SBATCH --ntasks-per-node=40  // specifies core per node
#SBATCH --time=06:50:20 // specifies maximum duration of run
#SBATCH --job-name=lammps               // specifies job name
#SBATCH --error=job.%J.err_node_40  // specifies error file name
#SBATCH --output=job.%J.out_node_40 //specifies output file name
#SBATCH --partition=standard          // specifies queue name
#SBATCH --nodelist=cn[031,046]        // nodelist specifies particular nodes
to be allocated
export I_MPI_FABRICS=shm:dapl
hostname
```

List Partition

sinfo displays information about nodes and partitions(queues).

$ sinfo



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

Figure 7 – Output of sinfo command

Submit the job

We can consider three cases of submitting a job

1. **Submitting a simple standalone job**

   This is a simple submit script which is to be submitted

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

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

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

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

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

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

```
$ sbatch -d singleton simple.sh  //may be used for first pre-processing
on a  core and then submitting
Submitted batch job 150
$ squeue
  JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
    150 standard   simple  user1  PD  0:00  1 (Dependency)
    149 standard   simple  user1   R   0:17  1 atom01
```

   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 atom01
```

3. **Submit a job with a reservation allocated**

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

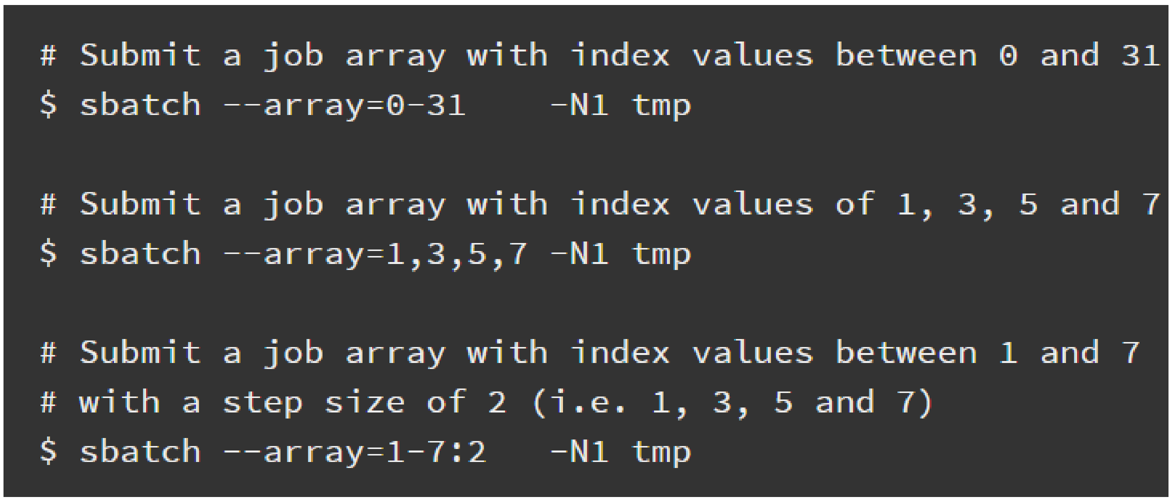
```
$ scontrol show reserv
```

   If your ‘user account’ is associated with any reservation the above command will show you the same. For eg. The reservation name given is user_11. Use below command 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.



```
# 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 8 – snapshot depicting the usage of “Job Array”

List jobs

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

```
$ squeue
 JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
   106 standard      slurm-jo  user1   R   0:04      1 atom01
```

Get job details

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

scontrol show node - shows detailed information about compute nodes.

scontrol show partition - shows detailed information about a specific partition

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

scontrol update job - change attributes of submitted job.

```
$ scontrol show job 106
JobId=106 Name=slurm-job.sh
   UserId=user1(1001) GroupId=user1(1001)
   Priority=4294901717 Account=(null) QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
   RunTime=00:00:07 TimeLimit=14-00:00:0 TimeMin=N/A
   SubmitTime=2013-01-26T12:55:02 EligibleTime=2013-01-26T12:55:02
   StartTime=2013-01-26T12:55:02 EndTime=Unknown
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=standard AllocNode:Sid=atom-head1:3526
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=atom01
   BatchHost=atom01
   NumNodes=1 NumCPUs=2 CPUs/Task=1 ReqS:C:T=*:*:*
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) Gres=(null) Reservation=(null)
   Shared=0 Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/user1/slurm/local/slurm-job.sh
   WorkDir=/home/user1/slurm/local
```

Suspend a job (root only):

```
# scontrol suspend 135
# squeue
  JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
    135  standard simple.s  user1  S   0:10   1    atom01
```

Resume a job (root only):

```
# scontrol resume 135
# squeue
  JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
    135  standard simple.s  user1  R   0:13   1    atom01
```

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

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

Hold a job:

```
$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    139      standard   simple  user1  PD       0:00      1 (Dependency)
    138      standard   simple  user1   R       0:16      1 atom01
$ scontrol hold 139
$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    139      standard   simple  user1  PD       0:00      1 (JobHeldUser)
    138      standard   simple  user1   R        0:32      1 atom01
```

Release a job:

```
$ scontrol release 139
$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    139      standard   simple  user1  PD       0:00      1 (Dependency)
    138      standard   simple  user1   R       0:46      1 atom01
```

Addressing Basic Security Concerns

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

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

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

More about Batch Jobs (SLURM)

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

It is important to note:

- Compilations are done on the login node. Only the execution is scheduled via SLURM on the compute/GPU nodes
- Upon Submission of a Job script, each job gets a unique Job Id. This can be obtained from the '**squeue**' command.
- The Job Id is also appended to the output and error filenames.

Parameters used in SLURM job Script

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

Resource	Flag Syntax	Description
partition	--partition=partition name	Partition is a queue for jobs.
time	--time=01:00:00	Time limit for the job.
nodes	--nodes=2	Number of compute nodes for the job.
cpus/cores	--ntasks-per-node=8	Corresponds to number of cores on the compute node.
resource feature	--gres=gpu:2	Request use of GPUs on compute nodes
account	--account=group-slurm-account	Users may belong to groups or accounts.
job name	--job-name="lammps"	Name of job.
output file	--output=lammps.out	Name of file for stdout.
email address	--mail-user=username@iitk.ac.in	User's email address
access	--exclusive	Exclusive access to compute nodes.

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 / Partition or queue name

// To load the module //
module load compiler/intel/2018.2.199

cd  <Path of the executable>.

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

Script for a Parallel OpenMP Job

```
#!/bin/bash
#SBATCH -N 1 / Number of nodes
#SBATCH --ntasks-per-node=24 / 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=standard / Partition or queue name

/ To load the module /
module load intel/2018.0.1.163
cd  < path of the executable>
Export OMP_NUM_THREADS=24 ( Depending upon your requirement you can change
number of threads . Maximum no.of threads is =24 )
/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=40 / 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_40 / Name of the output file
#SBATCH --output=job.%J.out_16_node_40 / Name of the error file
#SBATCH --partition=standard / Partition or queue name

// To load the module //
module load compiler/intel/2018.2.199

module unload gnu8/8.3.0

source
/opt/ohpc/pub/intel2018/compilers_and_libraries_2018.1.163/linux/mkl/bin/mk
lvars.sh intel64

// Below are the MPI Settings //

export I_MPI_FALLBACK=disable
export I_MPI_FABRICS=shm:tmi // Fabrics required for with node inter node
//
```

```
export I_MPI_DEBUG=9 // Level of debug //

cd /home/manjuv/LAMMPS_2018COMPILER/lammps-22Aug18/bench

//  Command to run the lammps in Parallel //

time mpiexec.hydra -genv I_MPI_DEBUG 9 -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=40 / 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_40 / Name of the output file
#SBATCH --output=job.%J.out_16_node_40 / Name of the error file
#SBATCH --partition=standard / Partition or queue name

// To load the module //
module load compiler/intel/2018.2.199

module unload gnu8/8.3.0

source
/opt/ohpc/pub/intel2018/compilers_and_libraries_2018.1.163/linux/mkl/bin/mk
lvars.sh intel64

// Below are the MPI Settings //

export I_MPI_FALLBACK=disable
export I_MPI_FABRICS=shm:tmi // Fabrics required for with node inter node
//
export I_MPI_DEBUG=9 // Level of debug //

Export OMP_NUM_THREADS=20 ( Depending upon your requirement you can change
number of threads . Maximum no.of threads is =40 )

cd /home/manjuv/LAMMPS_2018COMPILER/lammps-22Aug18/bench

//  Command to run the lammps in Parallel //

time mpiexec.hydra -genv I_MPI_DEBUG 9 -n $SLURM_NTASKS -genv
OMP_NUM_THREADS 20 /home/manjuv/LAMMPS_2018COMPILER/lammps-
22Aug18/src/lmp_intel_cpu_intelmpi -in in.lj
```

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

Environment Variables	PBS/Torque	SLURM
Job Id	$PBS_JOBID	$SLURM_JOBID
Submit Directory	$PBS_JOBID	$SLURM_SUBMIT_DIR
Node List	$PBS_NODEFILE	$SLURM_JOB_NODELIST
Job Specification	PBS/Torque	SLURM
Script directive	#PBS	#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]

Preparing your own Executable

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

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

Steps:

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

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

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

```
cp -r /home/apps/cdac/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/Fortran)	4.8.5, 8.3.0
icc/ifort	Intel Compiler (C/Fortran)	17.x, 18.x, 19x,20x
mpicc/mpif90	Intel-mpi based GNU compiler (C/Fortran)	
mpiicc/mpiifort	Intel-mpi based intel compiler (C/Fortran)	
nvcc	CUDA C Compiler	7.0,8.0,9.0,9.2,10.0,10.1
pgcc/pgf90	PGI Compiler (C/Fortran)	21.2, 2021

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 can be explored more on the respective compiler pages. A few examples are given below.

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

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

C Program:

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

C+OpenMP Program:

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

C+MPI Program:

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

C+MKL Program:

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

CUDA:

```
Setting up of environment: module load cuda/10.1
compilation: nvcc <<prog_name.cu>> -lcublas
Execution:./a.out
```

OpenACC:

```
Setting up of environment: module load compiler/nvhpc/21.2
Compilation: pgcc -acc -fast -Minfo=all -ta=tesla:managed -Mprof=ccff
<<prog_name.c>>
Execution:./a.out
```

Job Submission on Scheduler (SLURM):

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

A few sample commands for SLURM are as follows:

`Sinfo`	Lists out the status of resources in the system
`squeue`	Lists out the Job information in the system
`sbatch <<job_script>>`	Submitting a job to the scheduler

`scancel <<job_name>>`	Delete a job

Job Scheduling on PARAM Sanganak

Scheduler

PARAM Sanganak has Slurm-19.05.0-1 (open source) as a workload manager for HPC facility. Slurm is a widely used batch scheduler in top500 HPC list. PARAM Sanganak consists of three types of compute nodes: i.e. CPU only (192 GB) nodes, High memory (768 GB) nodes and nVIdia GPGPU (192 GB) enabled.

Following partitions/queues have been defined for different requirements

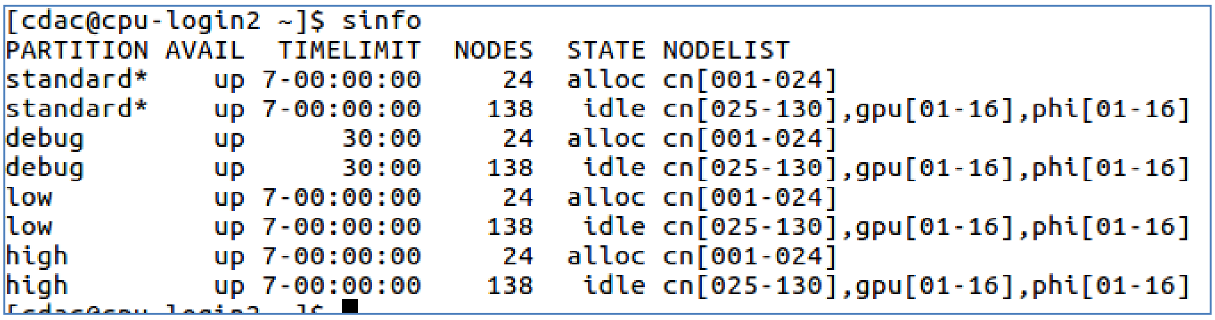
1. **standard**: CPU, High memory and GPU Jobs
2. **gpu**: GPU and CPU jobs
3. **hm:** High memory intensive jobs

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

Resource limits like priority etc., will be defined as per client requirements.

sinfo

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



```
[cdac@cpu-login2 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
standard*    up 7-00:00:00     24  alloc cn[001-024]
standard*    up 7-00:00:00    138   idle cn[025-130],gpu[01-16],phi[01-16]
debug        up      30:00     24  alloc cn[001-024]
debug        up      30:00    138   idle cn[025-130],gpu[01-16],phi[01-16]
low          up 7-00:00:00     24  alloc cn[001-024]
low          up 7-00:00:00    138   idle cn[025-130],gpu[01-16],phi[01-16]
high         up 7-00:00:00     24  alloc cn[001-024]
high         up 7-00:00:00    138   idle cn[025-130],gpu[01-16],phi[01-16]
```

Figure 9 – View of available partition and node information on the cluster

Walltime

Currently, the default wall time is 4 days. If more than 4 days are required, the wall time can be explicitly specified on command line as:

```
srun -t walltime <hours>
```

or as part of the submit script.

If a job requires more than 4 days, a special request needs to be sent to HPC coordinator and will be dealt with on a case by case basis.

Per user

- Every user will have quota of 500 GB in HOME file system (/home) and 1TB in SCRATCH file system(/scratch).
- 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.
- Other resource limits are like number of CPU nodes per user, number of running jobs at a time per user etc. will be defined as per the user requirements.

Scheduling Type

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

JOB PRIORITY

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

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

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

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

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

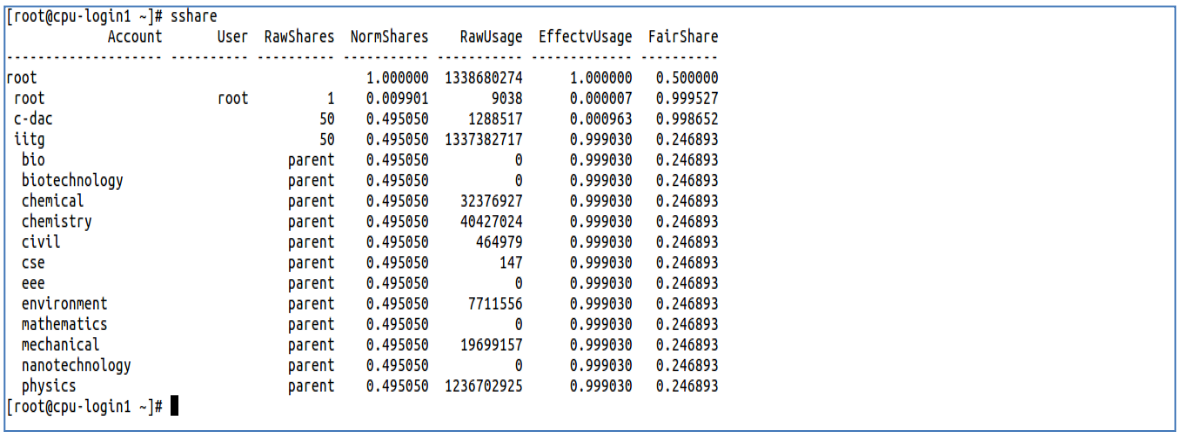
Partition Factor: Each node partition can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request to run in this partition. Current value for partition factor is 15000.

Quality of Service (QOS) Factor: Each QOS can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request this QOS. Current value for QOS factor is 1000.

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

SSHARE

This tool is for listing the shares of association to a cluster.



```
[root@cpu-login1 ~]# sshare
             Account       User  RawShares  NormShares    RawUsage  EffectvUsage  FairShare
-------------------- ---------- ---------- ----------- ----------- ------------- ----------
root                                          1.000000  1338680274      1.000000   0.500000
 root                      root          1    0.009901        9038      0.000007   0.999527
 c-dac                                  50    0.495050     1288517      0.000963   0.998652
 iitg                                   50    0.495050  1337382717      0.999030   0.246893
  bio                               parent    0.495050           0      0.999030   0.246893
  biotechnology                     parent    0.495050           0      0.999030   0.246893
  chemical                          parent    0.495050    32376927      0.999030   0.246893
  chemistry                         parent    0.495050    40427024      0.999030   0.246893
  civil                             parent    0.495050      464979      0.999030   0.246893
  cse                               parent    0.495050         147      0.999030   0.246893
  eee                               parent    0.495050           0      0.999030   0.246893
  environment                       parent    0.495050     7711556      0.999030   0.246893
  mathematics                       parent    0.495050           0      0.999030   0.246893
  mechanical                        parent    0.495050    19699157      0.999030   0.246893
  nanotechnology                    parent    0.495050           0      0.999030   0.246893
  physics                           parent    0.495050  1236702925      0.999030   0.246893
[root@cpu-login1 ~]#
```

Figure 10 - Listing the shares of association to a cluster.

ACCOUNTING

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

sacct

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

sstat

This command can be used to status only currently running jobs. It also can give you

sreport

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

Standard priority queue

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

High Priority queue

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

Storage Policy

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

Debugging Your Codes

Introduction

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

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

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

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

Basics How Tos

Compilation

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

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

Running with gdb:

gdb is a command line utility available with almost all Linux systems' compiler collection 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 command available. Specifying ‘help’ followed by a command (e.x. ‘help run’) displays more information about that command.

File <filename>:

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

List [line_no]

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

Info:

Displays more information about the set of utilities and saved information by the debugger. For example; ‘info breakpoints’ will list all the breakpoints, similarly ‘info watchpoints’ will list all the 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 function call (only function or instruction compiled with debugging flags).

next:

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

Continue:

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

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

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

watch <expression>:

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

Delete <breakpoint number>

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

Backtrace:

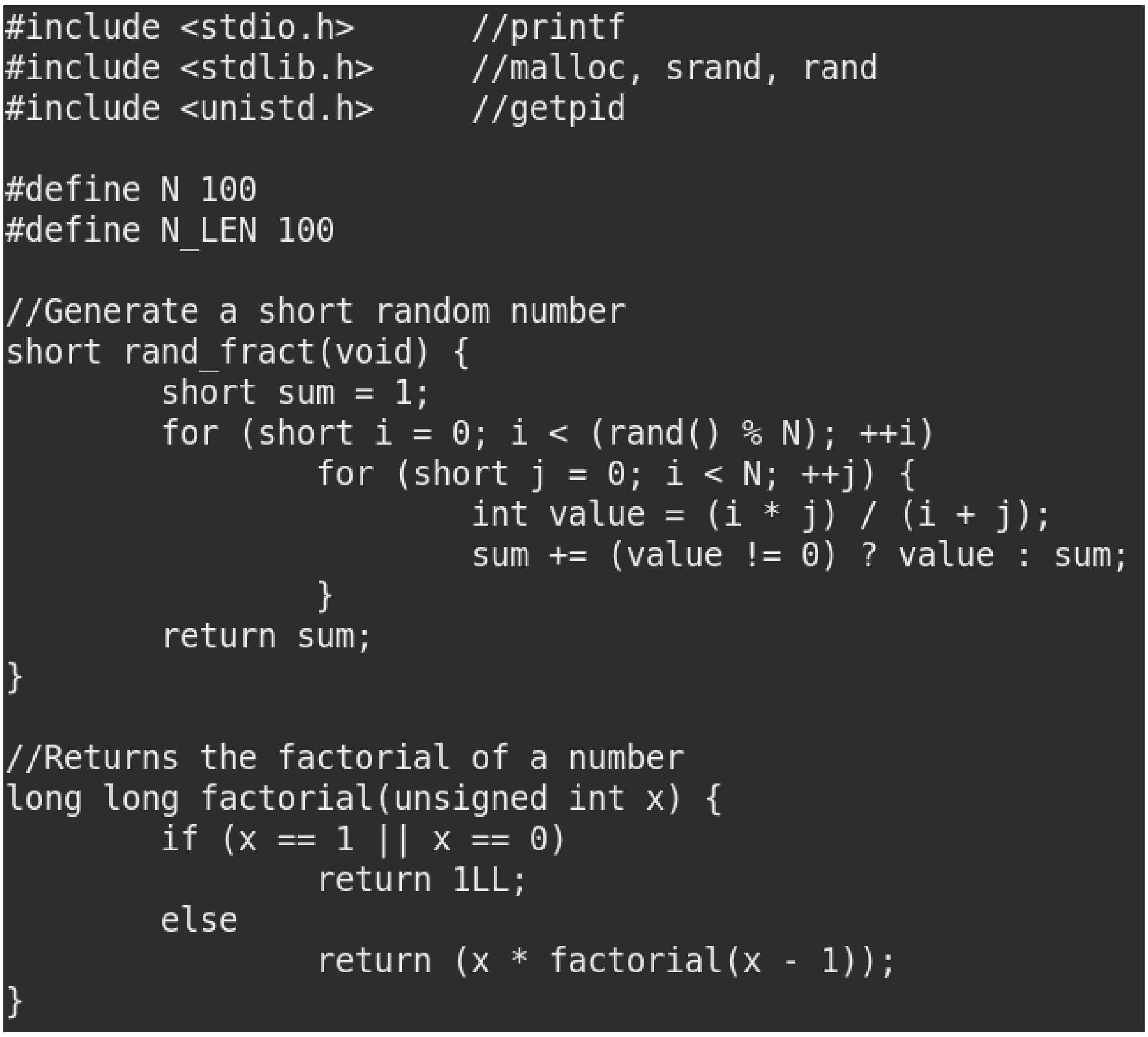
Prints the backtrace of all stack frames of the program. Provides the call stack and more other information about the running program.

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

Using gdb (example – inspecting the code)

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

Let's look at the code we have.



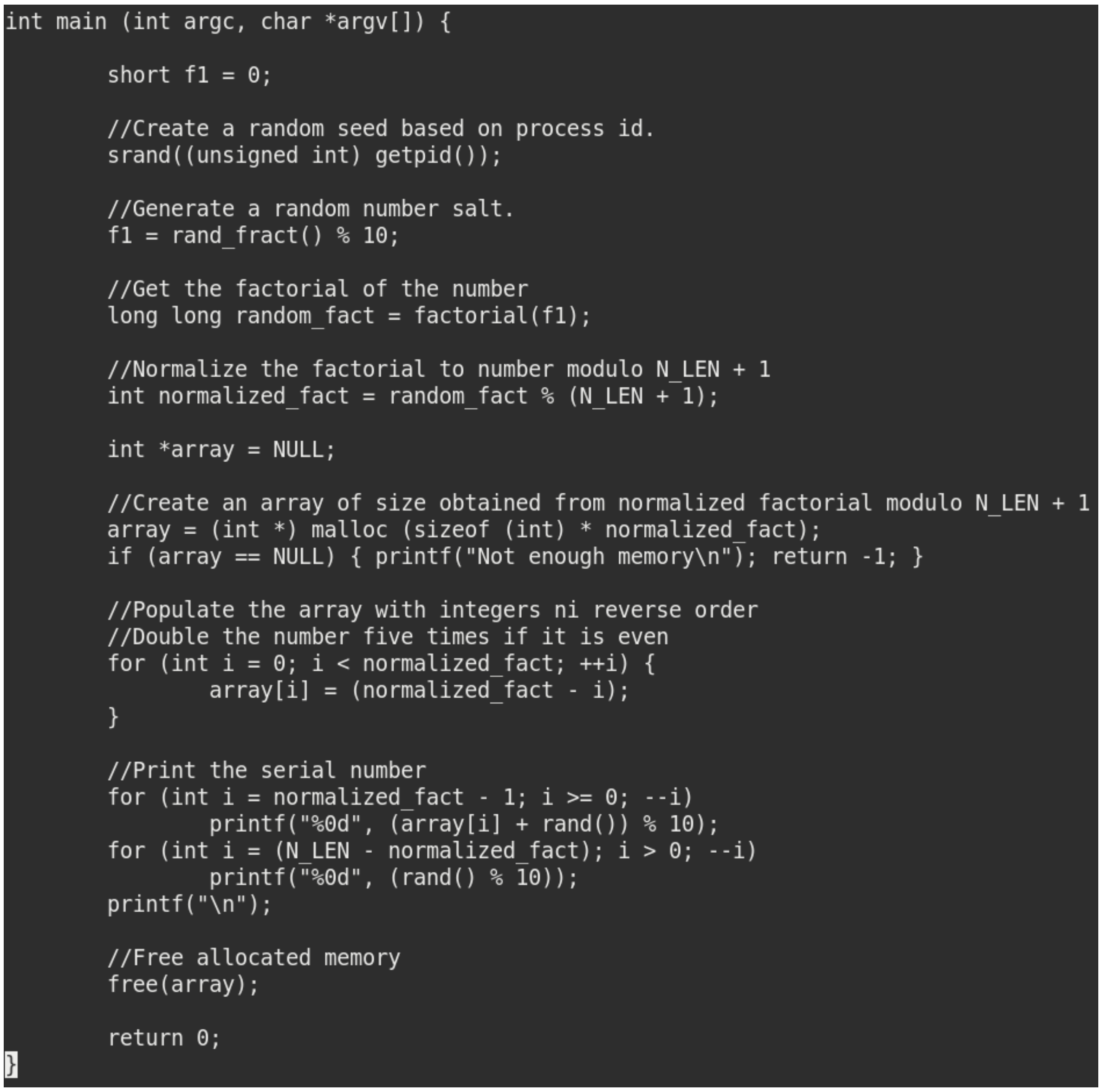
```
#include <stdio.h>      //printf
#include <stdlib.h>     //malloc, srand, rand
#include <unistd.h>     //getpid

#define N 100
#define N_LEN 100

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

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

Figure 11 – 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 12 – Snapshot of debugging process

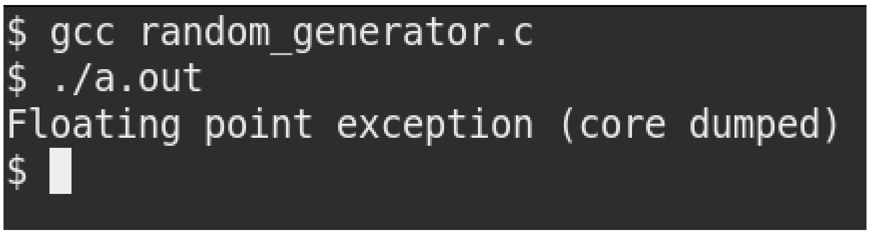
Things to note:

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 will integer values in descending order from the ‘normalized_fact’ value.

f. Finally, the partial array is printed by mixing the value of the array with rand() function values followed by a modulo 10 operation.
g. The remaining partial part of final random value is generated using a basic rand() modulo 10 operation.

Using gdb (example – using the debugger)

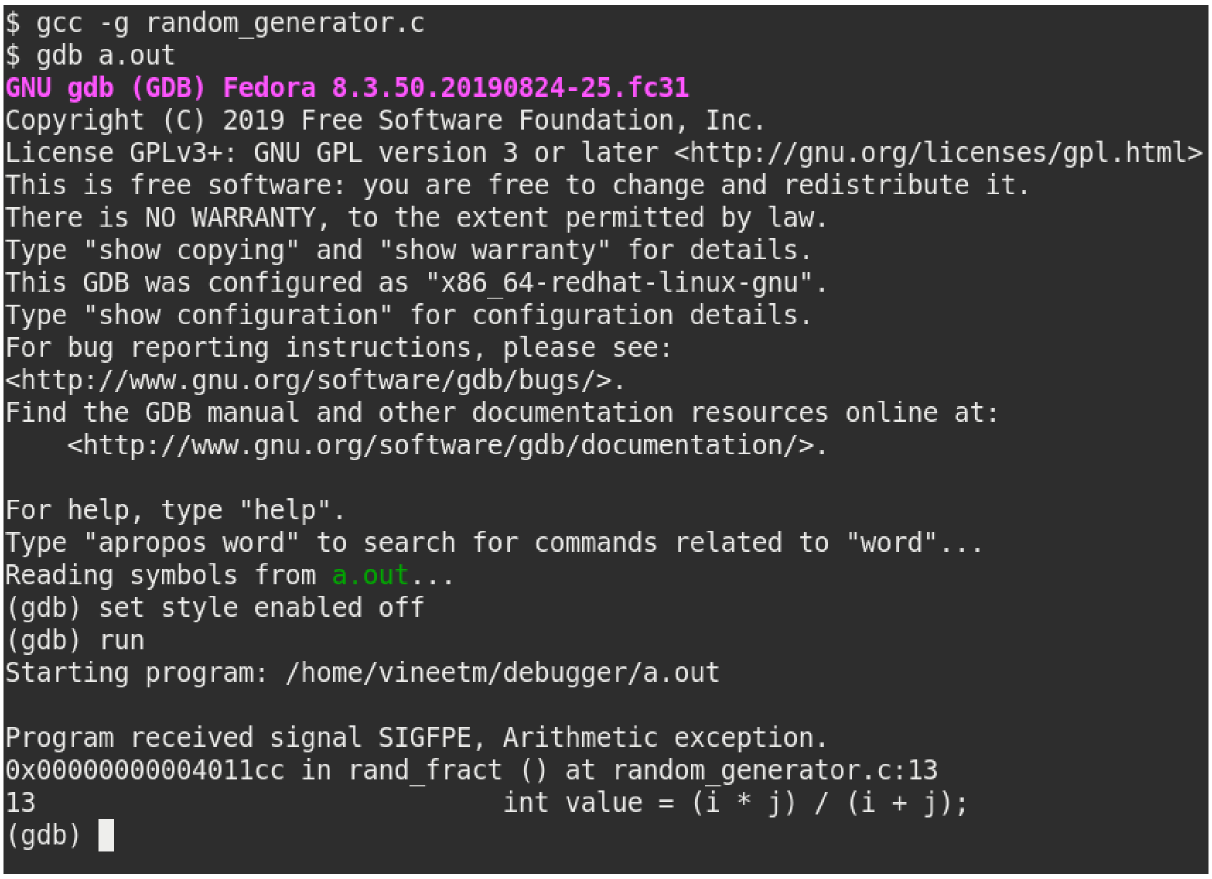
The code that we looked upon seems correct, as well as it compiles successfully without any errors. But, when we run this code snippet, this is the result we get.



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

Figure 13- output at a debugging stage

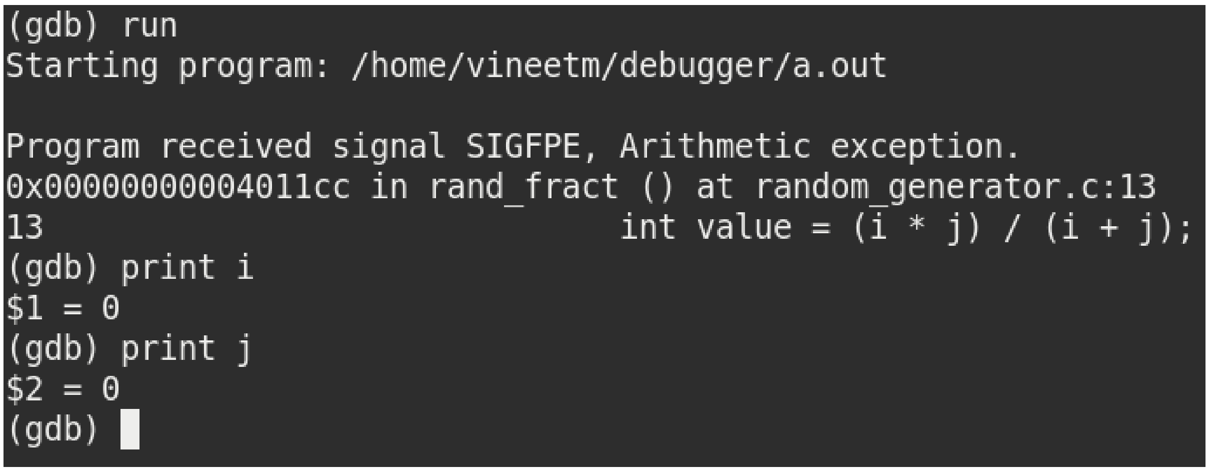
The program ended up with a core dump without giving much information but just 'Floating point exception'. Now let's compile the code with debugging information and run the program simply with gdb.



```
$ 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 14 – Snapshot of debugging process

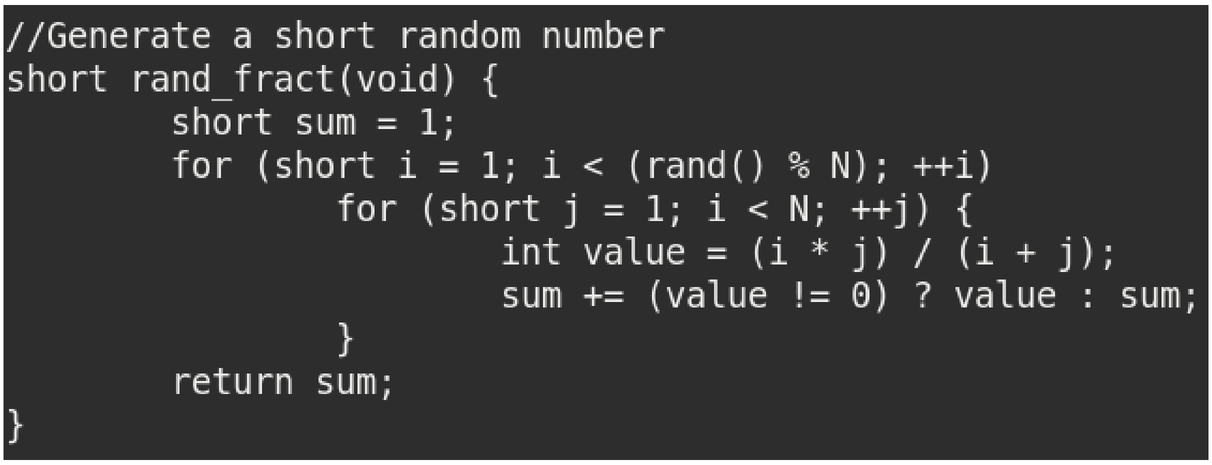
Here we compiled the code using '-g' and then used the 'run' command we studied earlier for running the program. You can observe that the debugger stopped at line number 13 where the 'Floating point exception (SIGFPE)' occurred. At this point we can even go and check the code at line number 13. But for 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 15 – Output depicting "Arithmetic Exception"

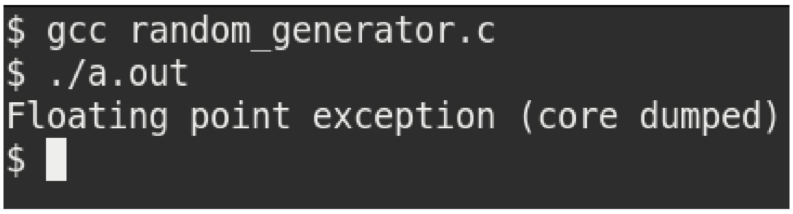
The values of both 'i' and 'j' appear to be '0' and thus a **divide by zero** exception is what caused our program to terminate. Let's update the code such that the value of 'i' and 'j' will never become '0'. This is the modified code:



```
//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 16 – Snapshot of debugging process

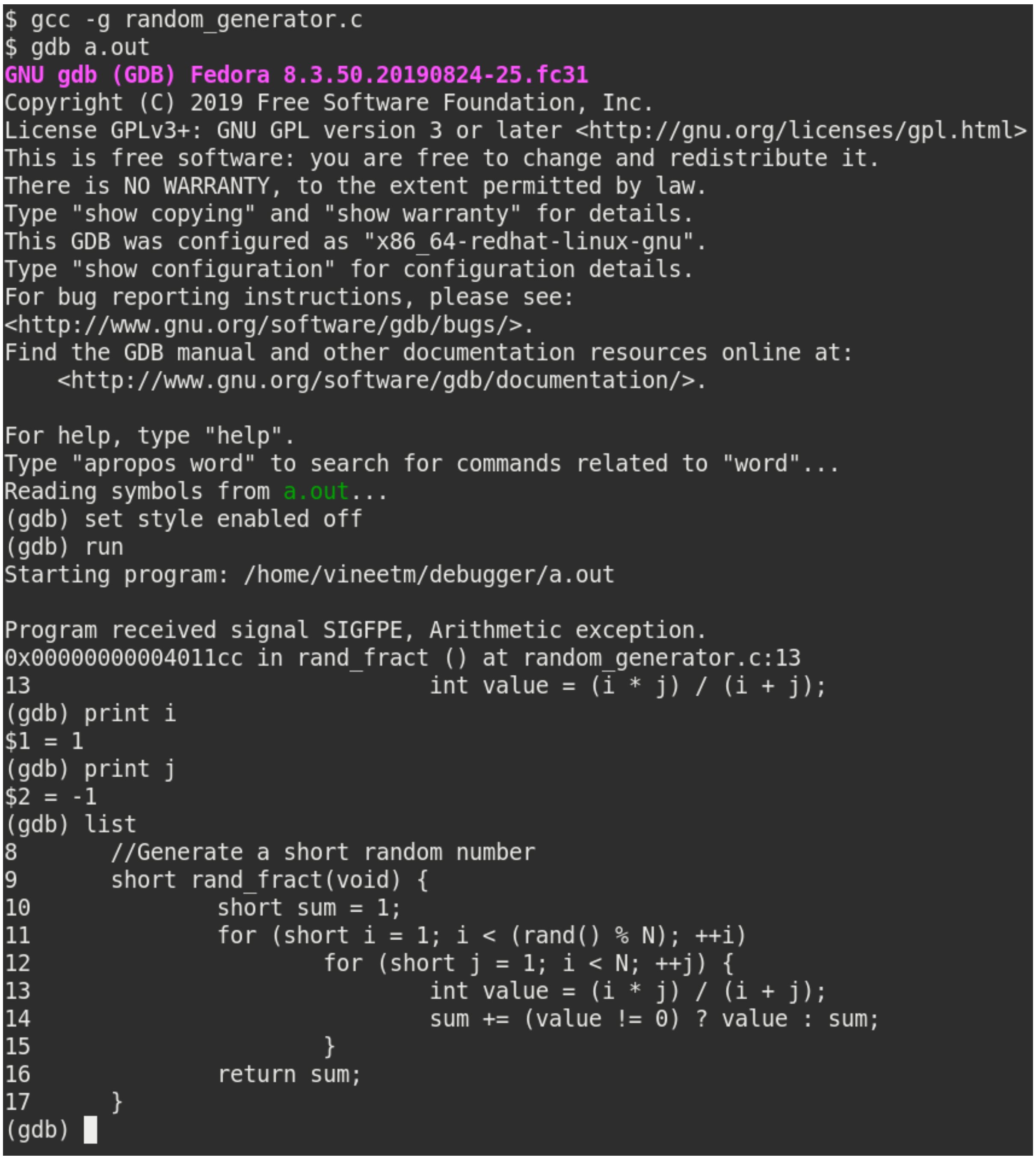
Thus, we just updated the loop index variables to start from '1' instead of '0'. **Thus, using gdb, it was very simple to identify the point where the error occurred.** Let's re-run our updated code and check what we get.



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

Figure 17 – 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.



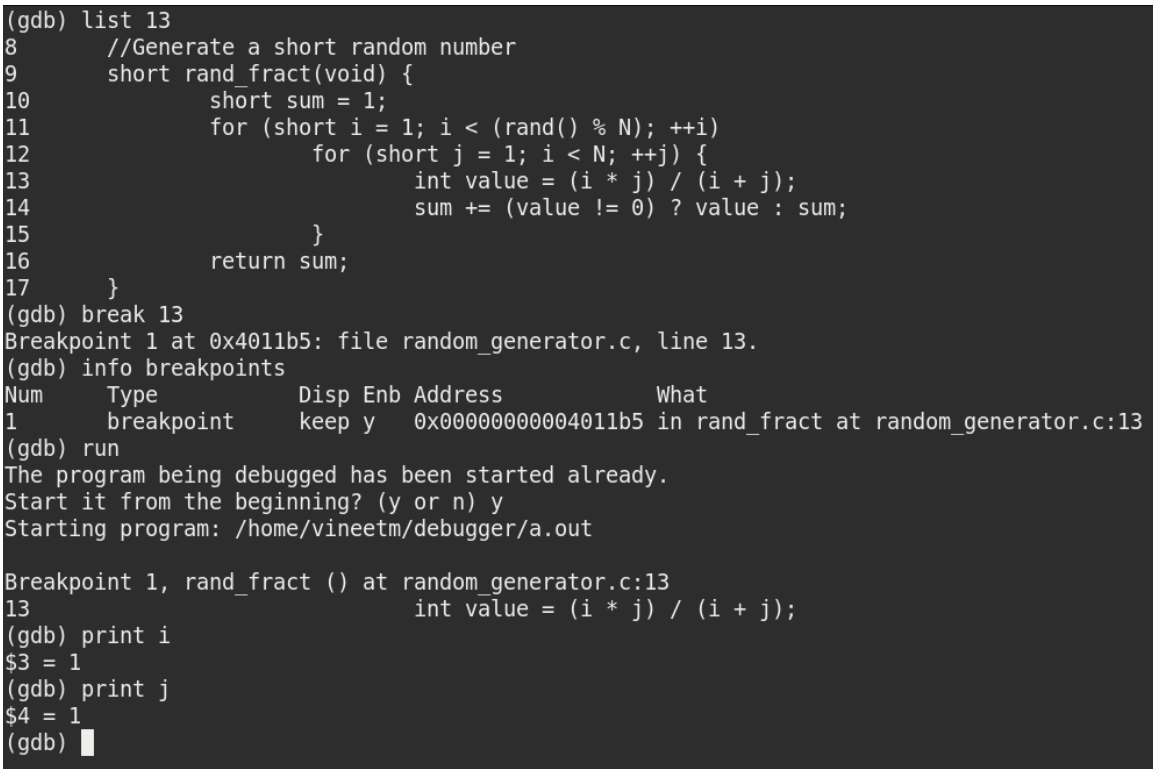
```
$ gcc -g random_generator.c
$ gdb a.out
GNU gdb (GDB) Fedora 8.3.50.20190824-25.fc31
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, 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 18 - 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.

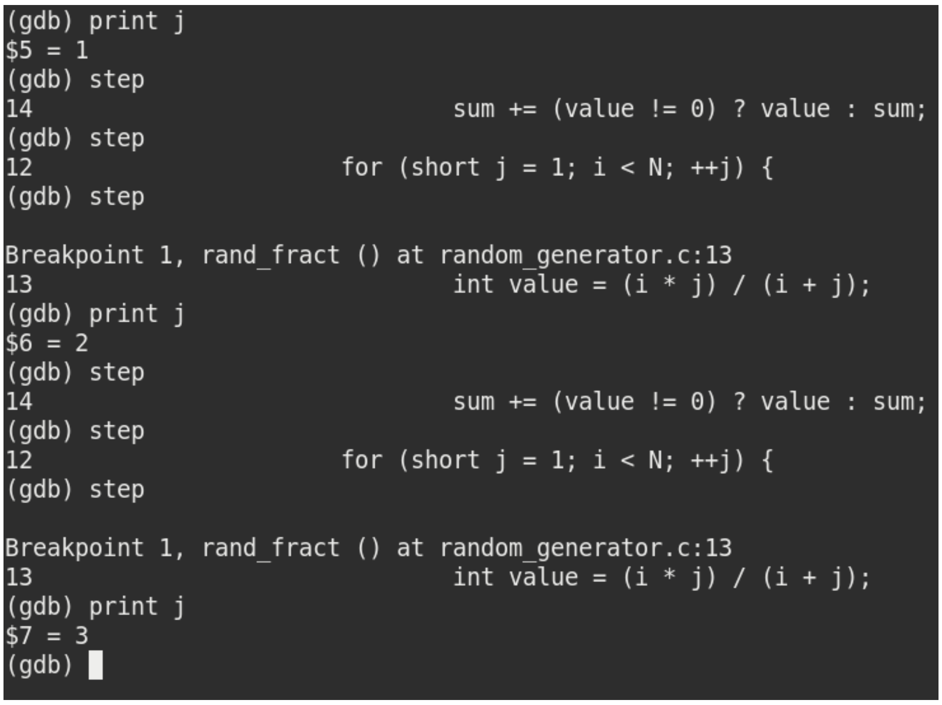
We will have to dig in much more to find out what is going on. Also, to be noted, we have our inner loop iterating from 1 to N (which is 100), but still the value of 'j' is printed out to be '-1'. How is this even possible!? Smart programmers would have the problem identified, but let's stick to the 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 19 – setting breakpoint

Thus, using the command 'break 13' we have set the breakpoint at line number 13 which was verified using the 'info breakpoint' command. Then, we reran the program with the 'run' command. At line 13 the program stopped and using 'print' command we checked the values of 'i' and 'j'. 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'.

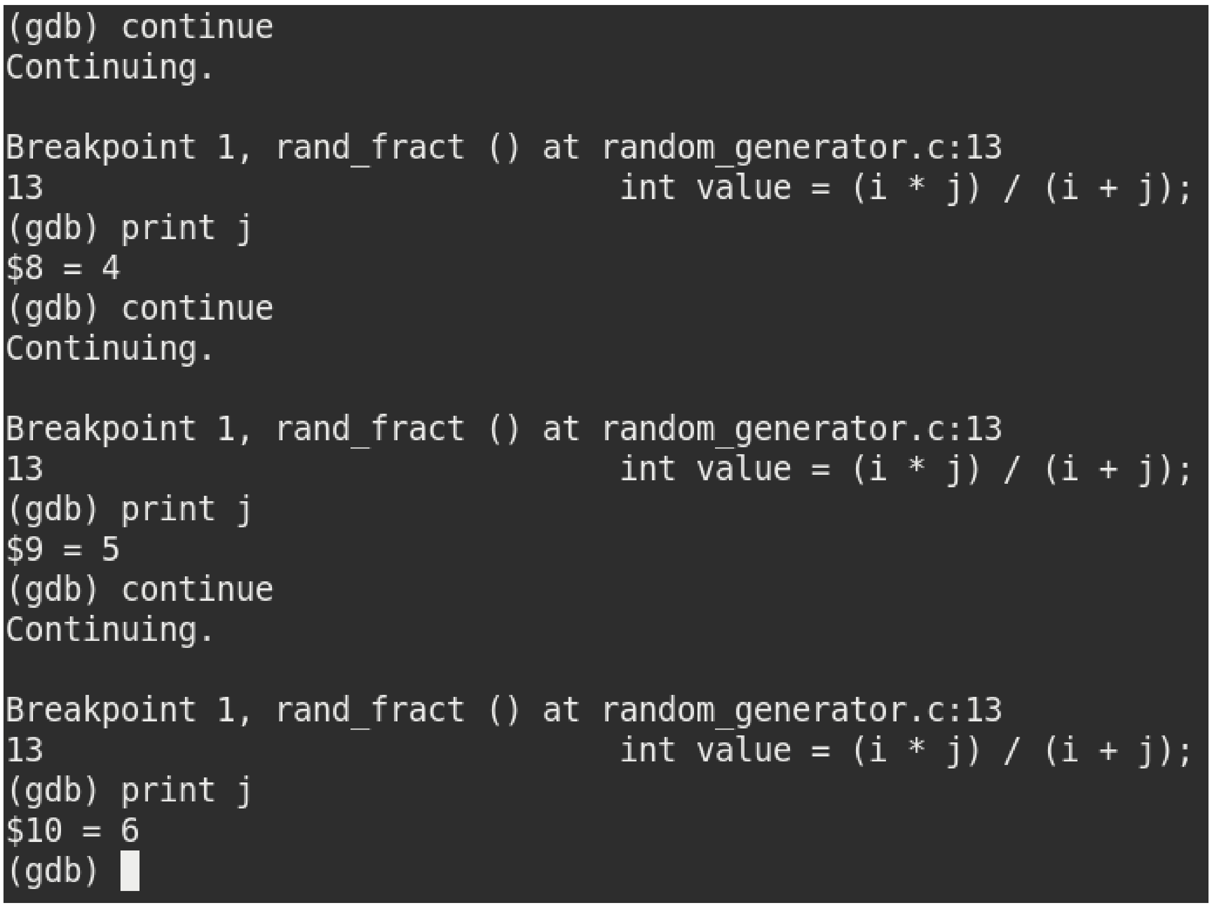


```
(gdb) print j
$5 = 1
(gdb) step
14                              sum += (value != 0) ? value : sum;
(gdb) step
12                      for (short j = 1; i < N; ++j) {
(gdb) step

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 20 – single stepping through to catch error !!

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

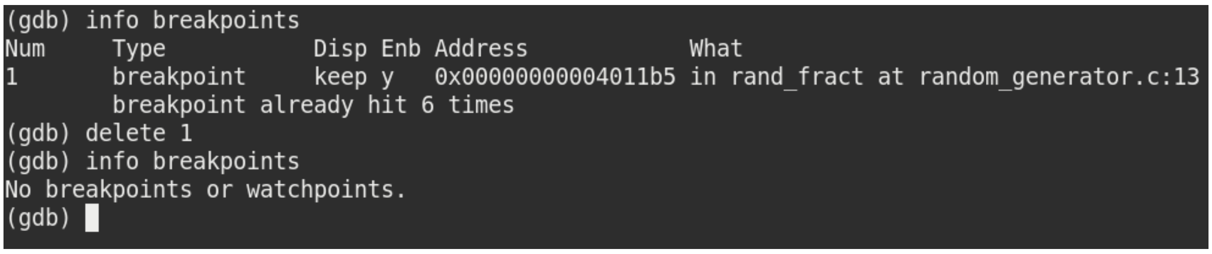
There seems to be a lot of writing/typing of the ‘step’ command just to proceed with the program. Since, we have already set a breakpoint at line 13, we can use another command called as ‘continue’. This command continues the program till the next breakpoint or the end of the program.



```
(gdb) continue
Continuing.

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

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

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

Figure 21 – Debugging continued

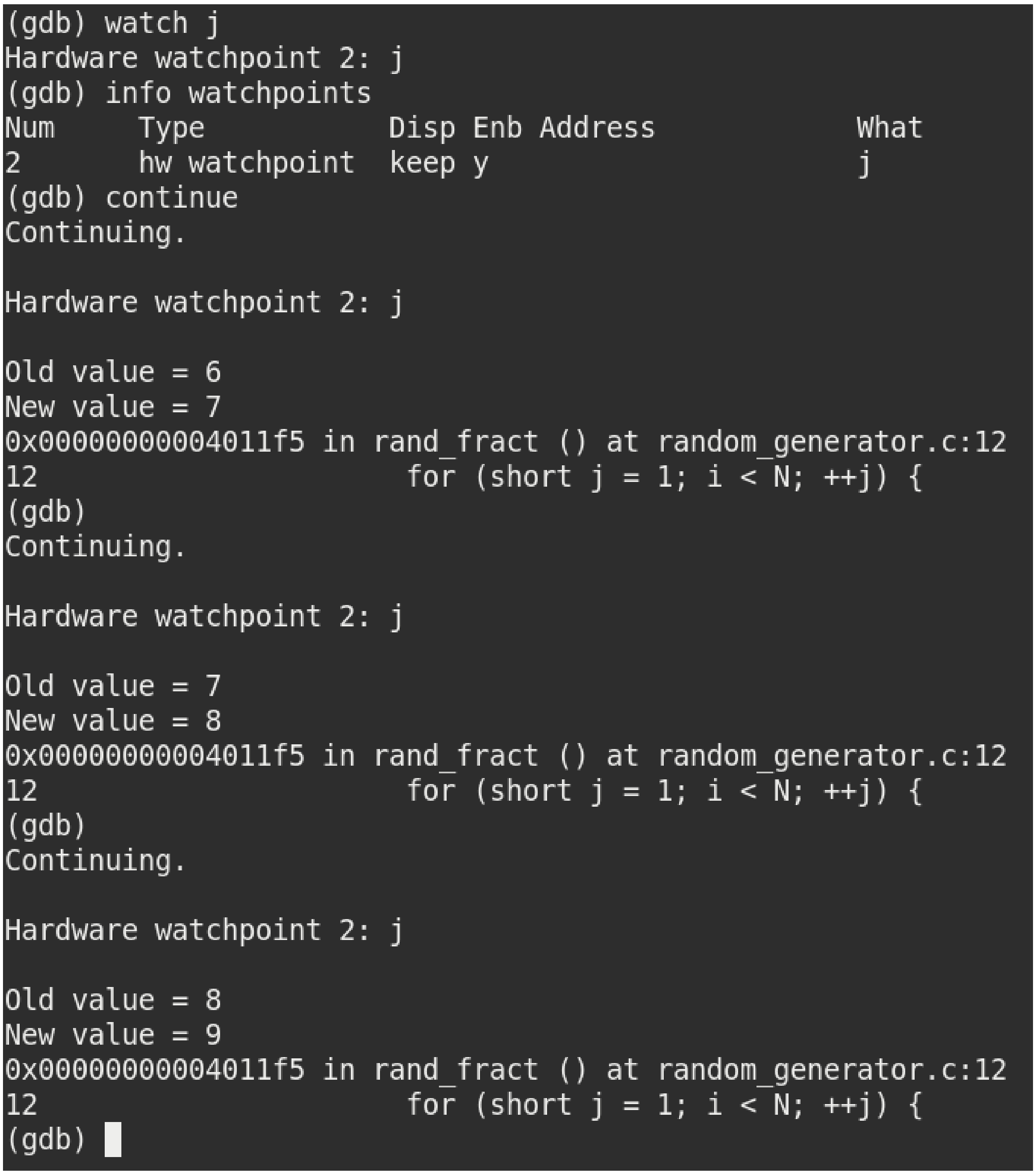
You can see that we reduced the typing of ‘step’ command by 3 times to a ‘continue’ command just 1 time. But this is also having us write ‘continue’ and ‘print’ multiple times. Let us use some other utility in gdb known as ‘data breakpoints’ also known as watchpoints. But before that, let us delete the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



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

Figure 22 – Debugging continued

Now let us see how to set a watchpoint.

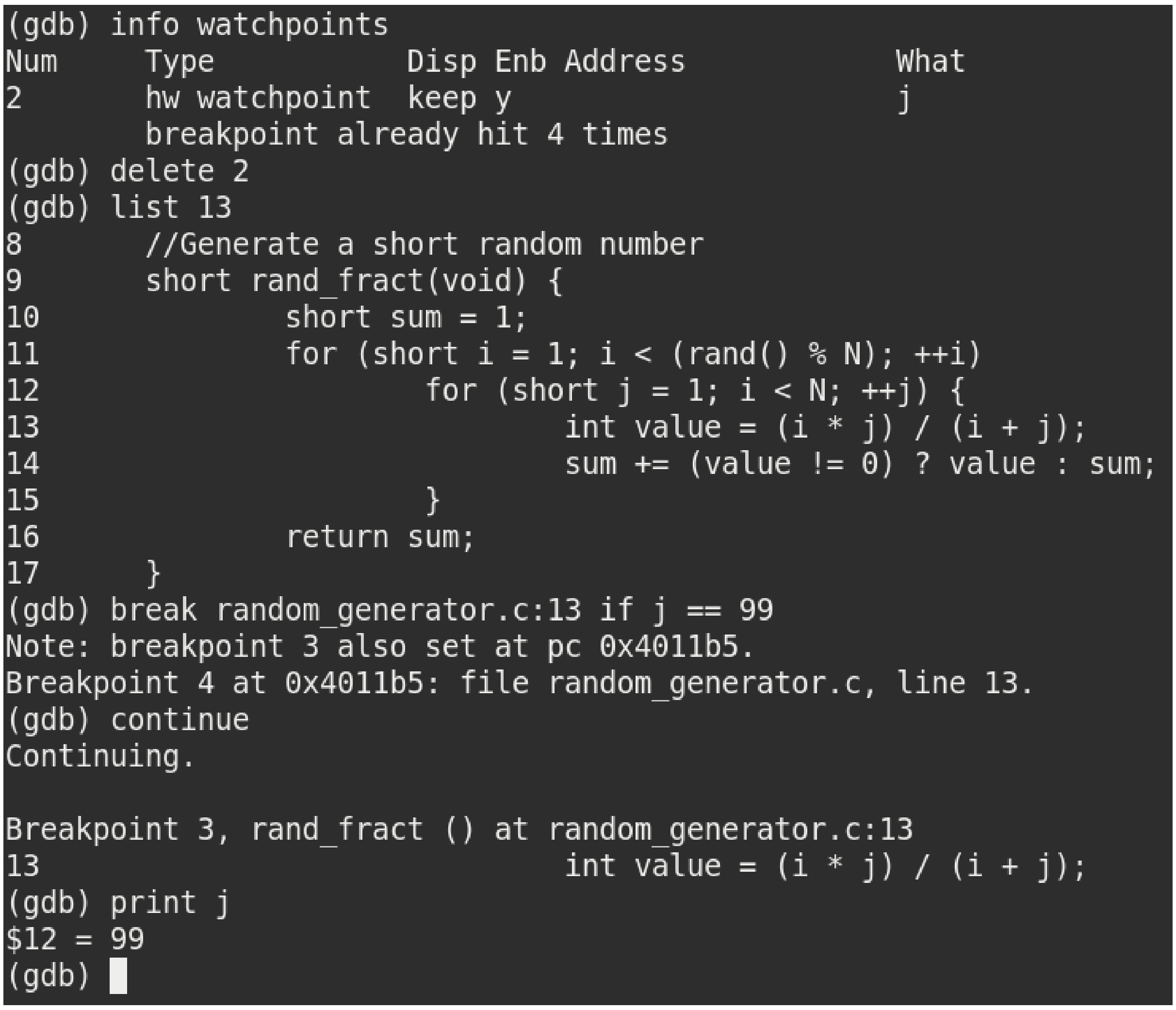


```
(gdb) watch j
Hardware watchpoint 2: j
(gdb) info watchpoints
Num     Type           Disp Enb Address            What
2       hw watchpoint  keep y                      j
(gdb) continue
Continuing.

Hardware watchpoint 2: j

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

Hardware watchpoint 2: j

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

Hardware watchpoint 2: j

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

Figure 23 – Setting a watch point

Thus, using the command ‘watch j’ we have set a watchpoint over ‘j’. Now every time when the value of ‘j’ changes, a break will occur. You can also note the old and new values of ‘j’ printed out at each break. Another point to note is that after having one ‘continue’ command, the program had a break. Further, by just pressing the ‘Enter/Return’ button on the keyboard, the continue command was repeated. Thus, by pressing the ‘Enter/Return’ button, the last command is repeated. At this point, we have learned much about the debugger, but we are still not able to proceed fast with our error. Is there any other way to procced? Well, yes!!

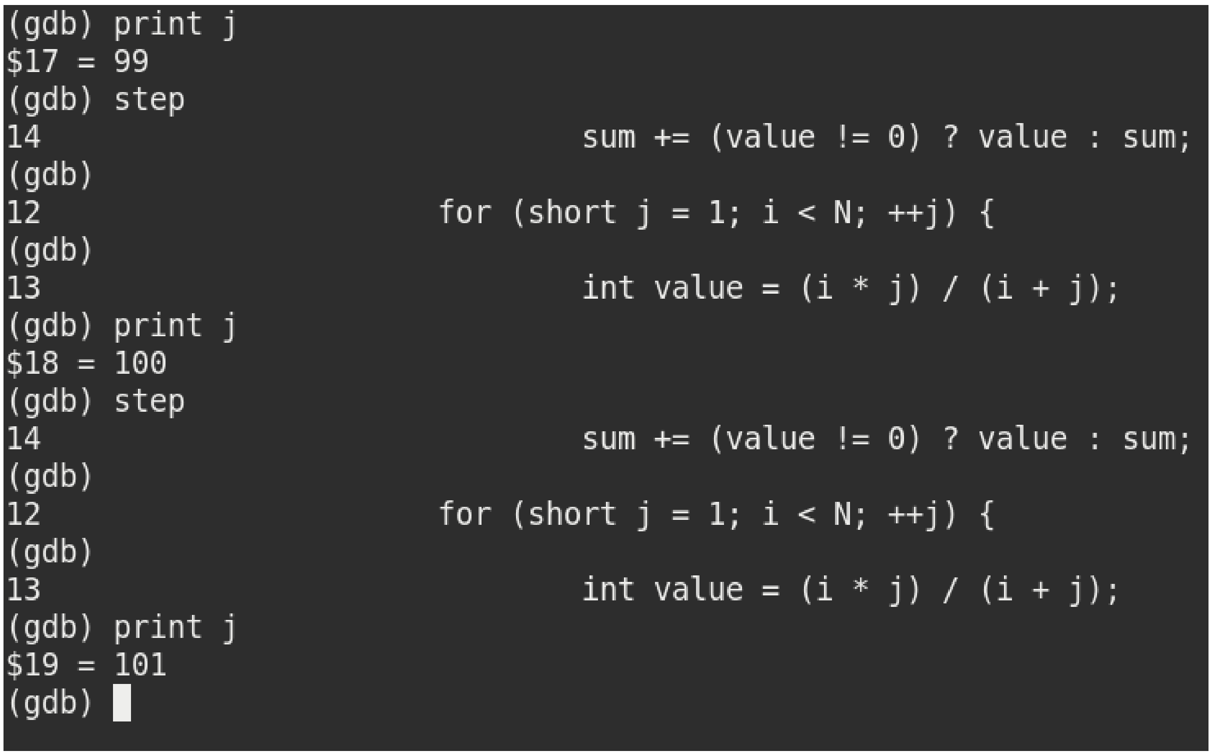
We want to observe at the point where the value of ‘j’ reaches closer to ‘N i.e. 100’. Which means that we are only concerned about what happens after ‘j’ reaches 99. Here, we land up on using what is called as conditional breakpoints. First, we will delete our watchpoint and then make use of the conditional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



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

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

Figure 24 – Debugging continued

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



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

Figure 25 – Well, Back to square one !!

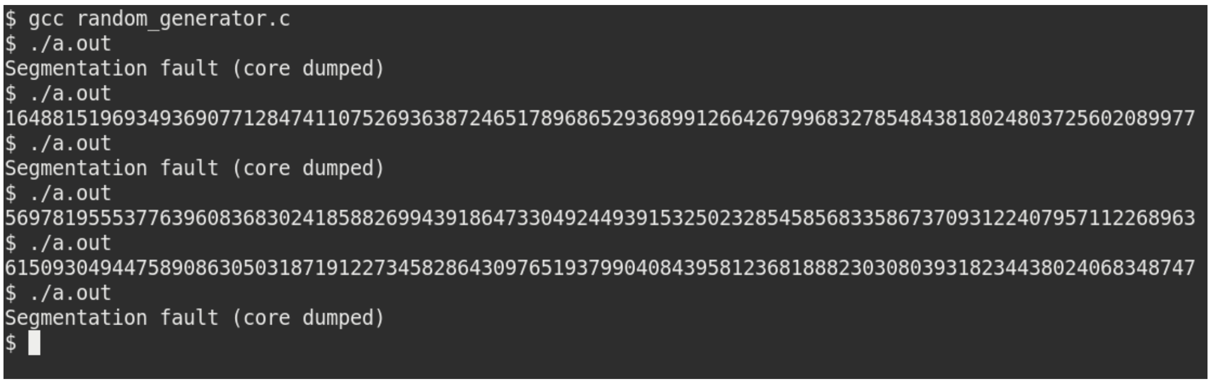
This, is unexpected!! The value of ‘j’ should never be 100 or anything above it.

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 lead us to this much of an effort.

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

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

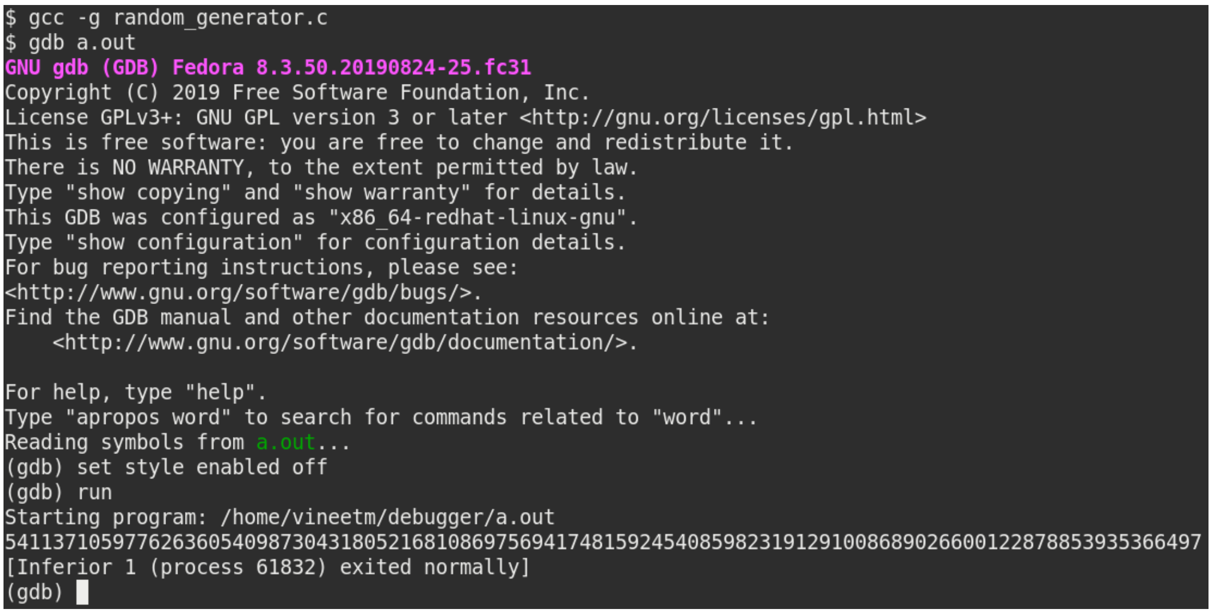


```
$ gcc random_generator.c
$ ./a.out
Segmentation fault (core dumped)
$ ./a.out
1648815196934936907712847411075269363872465178968652936899126642679968327854843818024803725602089977
$ ./a.out
Segmentation fault (core dumped)
$ ./a.out
5697819555377639608368302418588269943918647330492449391532502328545856833586737093122407957112268963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

Figure 26 – Again Dumping Core!! Things are getting interesting or frustrating or both !!

This is strange!!

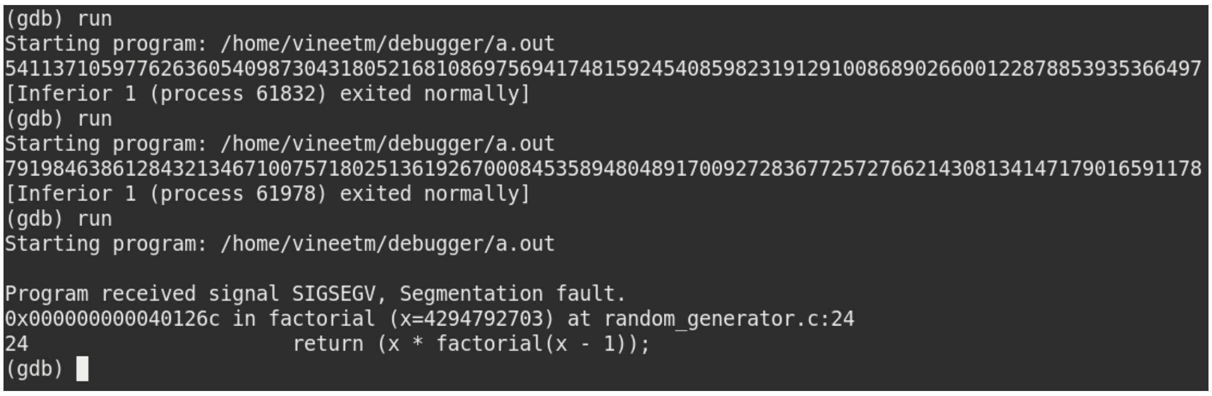
Sometimes the program is getting the correct output, but sometimes, we are getting a segmentation fault. Debugging such a program may be tricky since the occurrence of the bug is low. We will proceed with our standard debugger steps to identify the error.



```
$ 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 27 – 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.



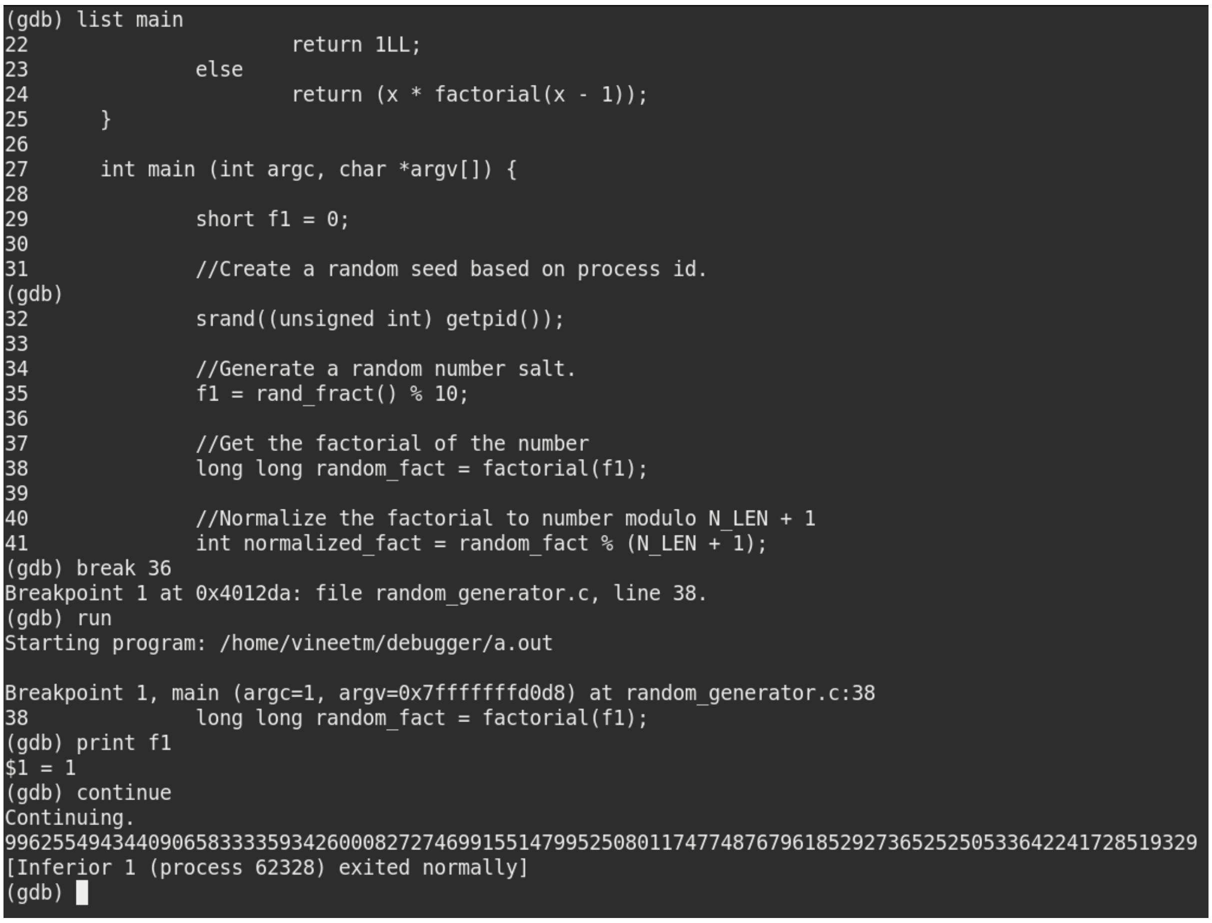
```
(gdb) run
Starting program: /home/vineetm/debugger/a.out
5411371059776263605409873043180521681086975694174815924540859823191291008689026600122878853935366497
[Inferior 1 (process 61832) exited normally]
(gdb) run
Starting program: /home/vineetm/debugger/a.out
7919846386128432134671007571802513619267000845358948048917009272836772572766214308134147179016591178
[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 28 – 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!

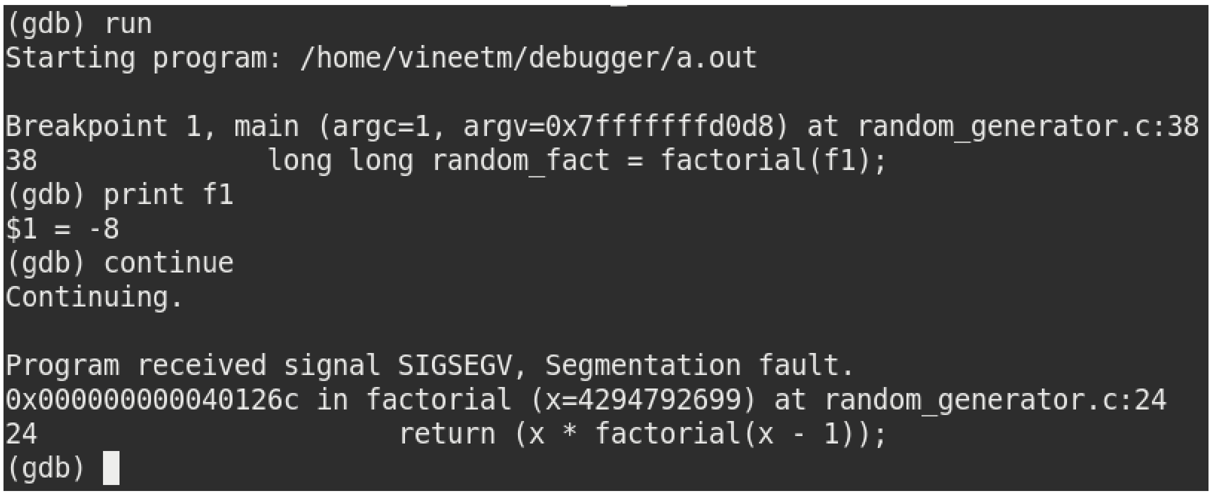
Another reason for such kind of output would be because of the recursive nature of the function. The stack frame where the function ‘factorial’ failed could be in a long nest of recursive calls. At such points, it would be better to inspect the program at an earlier point and look for errors. Let us 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 29 – Debugging continued (Will it ever end?)

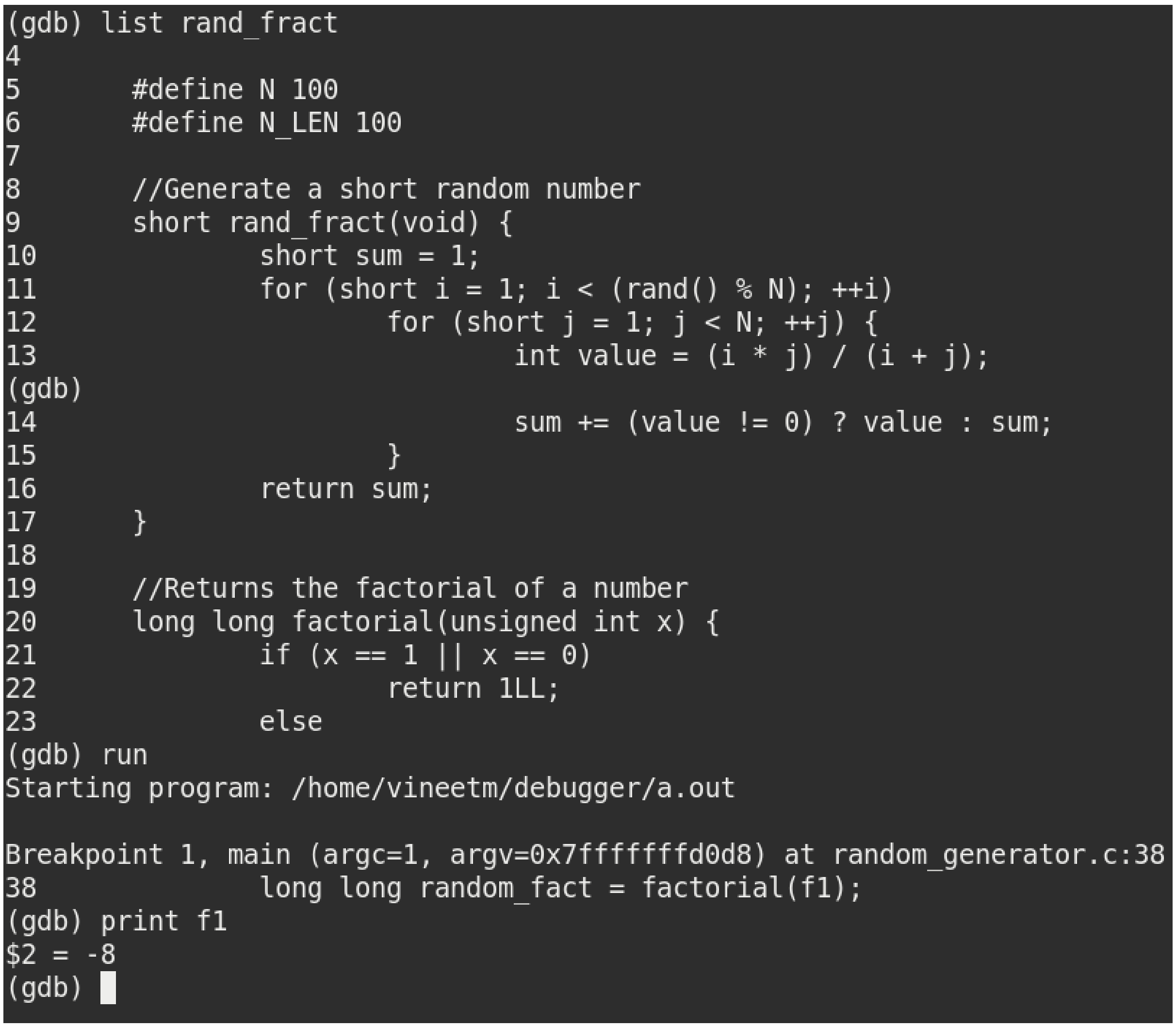
Thus, we have set a breakpoint before the call of the function ‘factorial’ and ran the program. For the value of ‘f1 = 8’ for the ‘factorial’ function the process seems to exit normally. Let us rerun.



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

Breakpoint 1, main (argc=1, argv=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 30 – 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 31 – Debugging continued

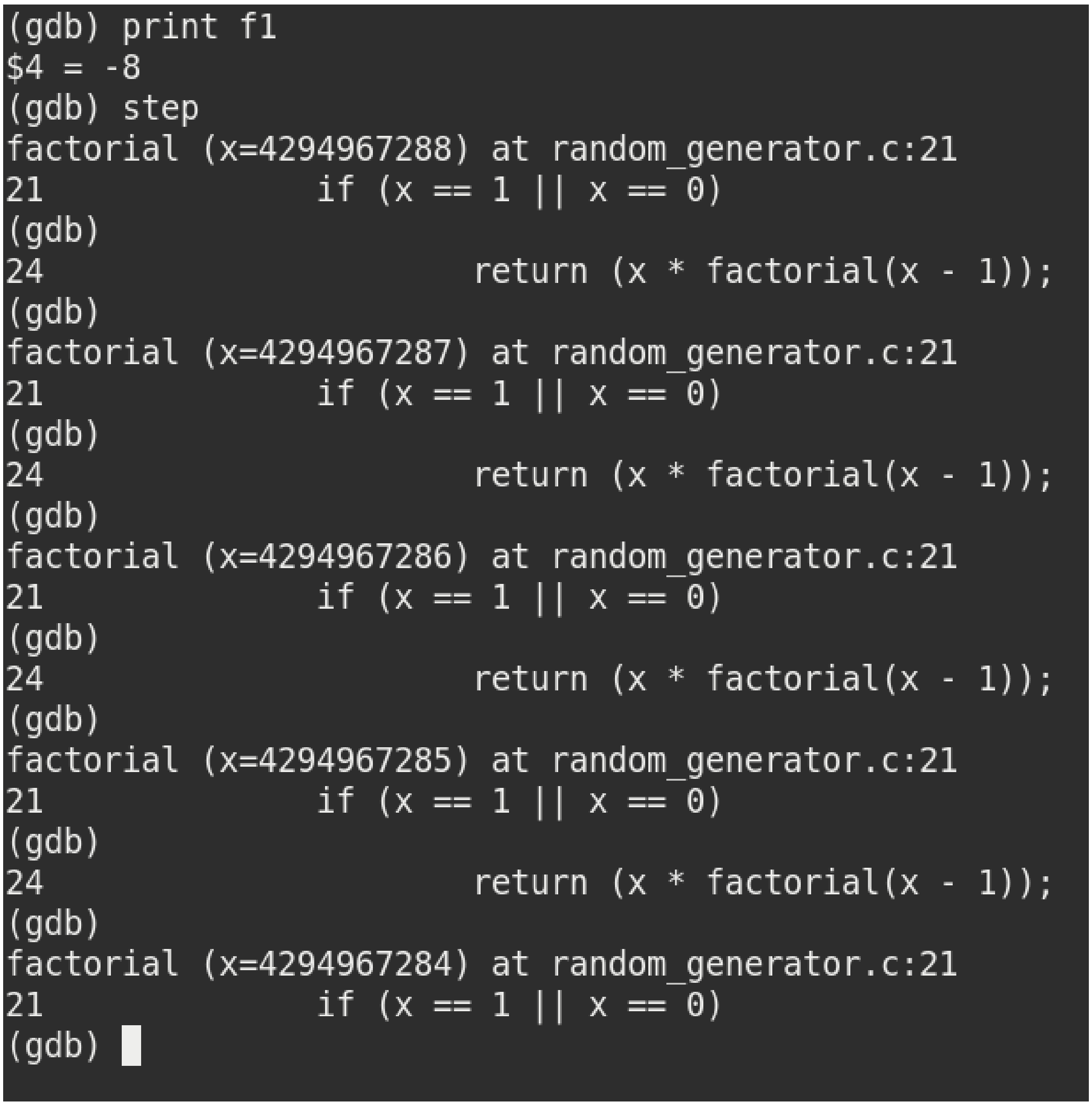
Important points here to observe are:

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 32 – At last a clue!!!

This is what we had expected!!
A number '-1' passed to the 'factorial' function is being implicitly converted to a very large number '4294967295'.

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

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

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

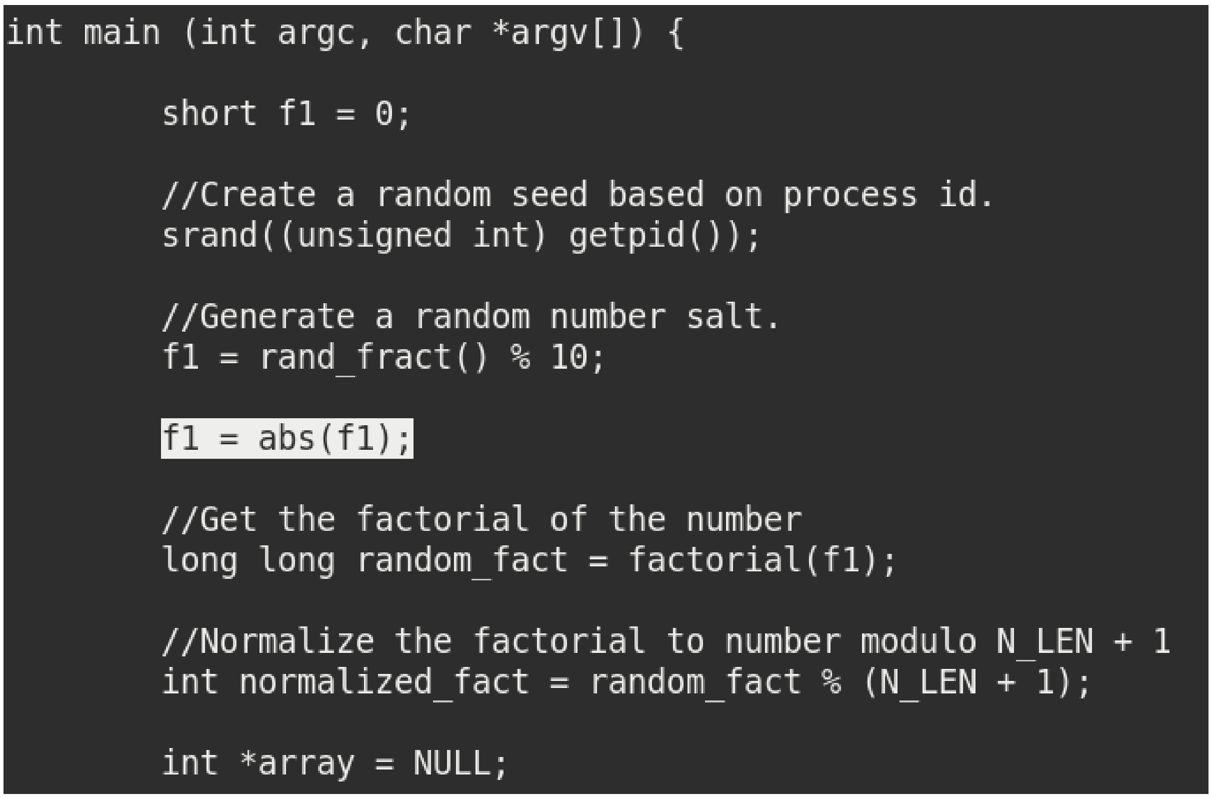
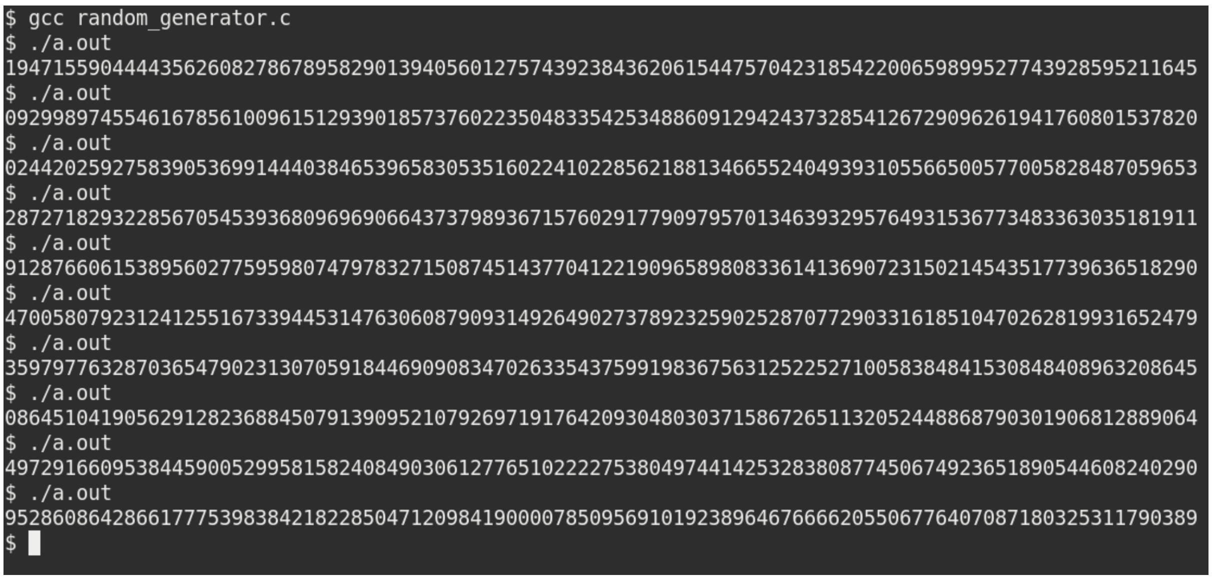

```
int main (int argc, char *argv[]) {

        short f1 = 0;

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

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

        f1 = abs(f1);

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

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

        int *array = NULL;
```

Figure 33 - Correction applied !!

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

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



```
$ gcc random_generator.c
$ ./a.out
1947155904444356260827867895829013940560127574392384362061544757042318542200659899527743928595211645
$ ./a.out
0929989745546167856100961512939018573760223504833542534886091294243732854126729096261941760801537820
$ ./a.out
0244202592758390536991444038465396583053516022410228562188134665524049393105566500577005828487059653
$ ./a.out
2872718293228567054539368096969066437379893671576029177909795701346393295764931536773483363035181911
$ ./a.out
9128766061538956027759598074797832715087451437704122190965898083361413690723150214543517739636518290
$ ./a.out
4700580792312412551673394453147630608790931492649027378923259025287077290331618510470262819931652479
$ ./a.out
3597977632870365479023130705918446909083470263354375991983675631252252710058384841530848408963208645
$ ./a.out
0864510419056291282368845079139095210792697191764209304803037158672651132052448868790301906812889064
$ ./a.out
4972916609538445900529958158240849030612776510222275380497441425328380877450674923651890544608240290
$ ./a.out
9528608642866177753983842182285047120984190000785095691019238964676666205506776407087180325311790389
$ █
```

Figure 34 – Resolved !!!

Thus, we are getting the correct results as expected.

Conclusions

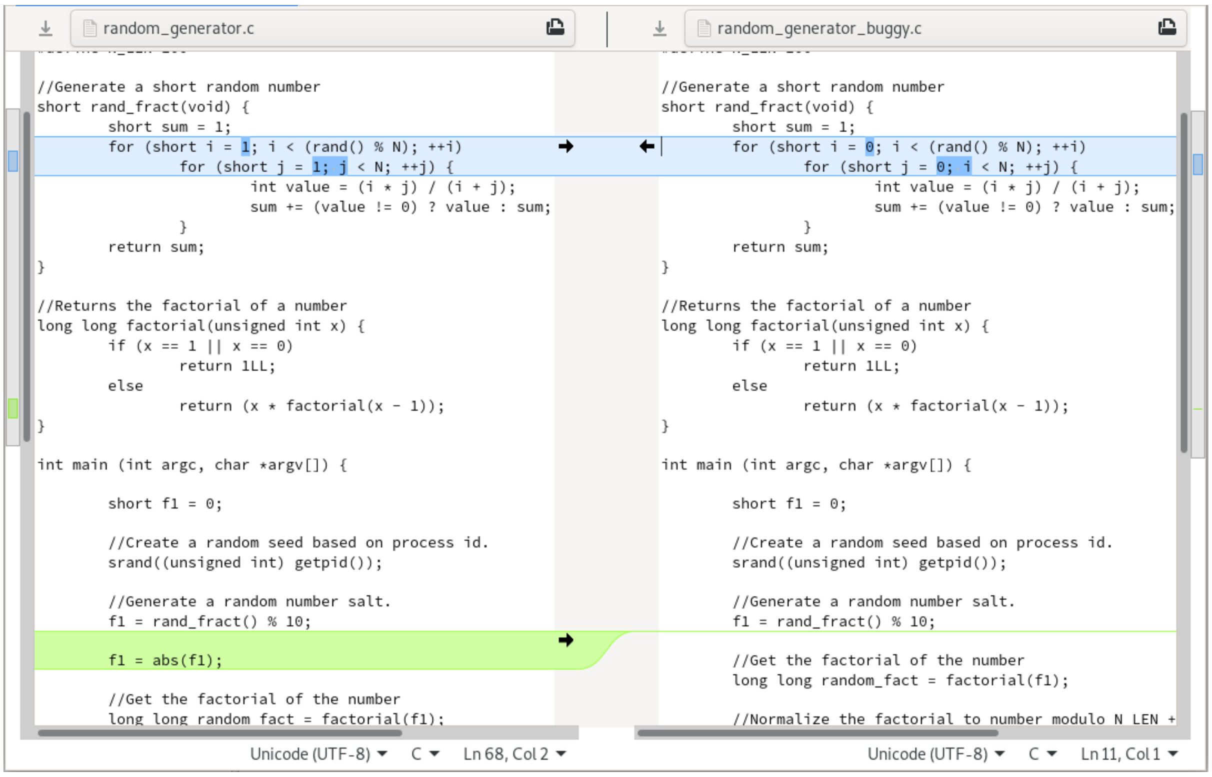
We started with a program that we assumed to be functional but then the program ended up with bugs that were not straightforward. We then explored the power of the debugger and the various ways to identify the bugs in our program. We looked upon the easy solutions, and slowly migrated towards the type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

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

Points to Note

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

Overall Coding Modifications Done



random_generator.c

```c
//Generate a short random number
short rand_fract(void) {
        short sum = 1;
        for (short i = 1; i < (rand() % N); ++i)
                for (short j = 1; j < 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));
}

int main (int argc, char *argv[]) {

        short f1 = 0;

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

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

        f1 = abs(f1);

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

Unicode (UTF-8) C Ln 68, Col 2

random_generator_buggy.c

```c
//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));
}

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 +
```

Unicode (UTF-8) C Ln 11, Col 1

Figure 35 – What all we did to get things right!

Machine Learning / Deep Learning Application Development

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

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

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

```
      $ module list
```

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

```
      $ module avail
```

To activate conda environment on PARAM Sanganak, load module "conda-python/3.7" as shown below:

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

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

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

Once "conda-python/3.7" module is loaded, end-users can use all libraries inside their python program. Many other modules based on virtual env are available on the system.

Users can load those libraries using “module load” command and use them for their applications.

How to Install your own Software?

There are two approaches to install software.

1. System wide installation
2. Local installation.

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

Local installation

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

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

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

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

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

Step 4. go to the software folder.

```
$ cd hmmer-3.3
```

Step 5. configure the installation path.

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

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

```
$ make
```

The newly compiled binaries are now in the src directory.

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

```
$ make check
```

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

```
$ make install
```

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

* This is general instruction for installation, please refer the installation instruction or manual or readme file that comes with software for more details.

# if you get any dependency error, resolve that or ask system admin to install that dependency if not installed.

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

Some Important Facts

About File Size

The global/home is served by a number of storage arrays. Each of the storage array contains a portion of the global/home. The size of a disk in the storage array is 2TB (2000 GB). Technically, the size of a file can be about 2000 GB (which is really big). However, since the disk is shared by a large number of files, effectively the size of a single file will be far smaller. Normally, this file size is kept to be about few GBs which is sufficient for most of the users. However, if you wish to have file sizes which are larger than this, you need to create files ACROSS disks and this process is known as 'striping'.

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

After this has been done all new files created in the current directory will be spread over 4 storage arrays each having 1/4th of the file. The file can be accessed as normal no special action needs to be taken. When the striping is set this way, it will be defined on a per directory basis so different directories can have different stripe setups in the same file system, new 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 are also able to set stripe size and stripe count for their directories and A user can check the set stripe size and stripe count with command:

```
lfs getstripe <path to the 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 of another machine (with a different processor).

* * *

Best Practices for HPC

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

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

LAMMPS Applications

LAMMPS is an acronym for **L**arge-scale **A**tomic/ **M**olecular **M**assively **P**arallel **S**imulator. This is extensively used in the fields of Material Science, Physics, Chemistry and may others. More information about LAMMPS may please be found at https://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 2
#SBATCH --ntasks-per-node=40
#SBATCH --time=02:50:20
#SBATCH --job-name=lammps
#SBATCH --error=job.%J.err_2_node_40
#SBATCH --output=job.%J.out_2_node_40
#SBATCH --partition=standard
#SBATCH --exclusive

module unload gnu8/8.3.0
module load intel/2018.2.199


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


export I_MPI_FALLBACK=disable
export I_MPI_FABRICS=shm:dapl
export I_MPI_DEBUG=9

cd /home/manjunath/LAMMP_TEST/LAMMPS_2018/lammps-22Aug18/bench

export OMP_NUM_THREADS=1


time mpiexec.hydra -n $SLURM_NTASKS -genv OMP_NUM_THREADS 1
/home/manjunath/LAMMP_TEST/LAMMPS_2018/lammps-
22Aug18/src/lmp_intel_cpu_intelmpi -in in.lj
```

3. LAMMPS OUTPUT FILE.

```
LAMMPS (22 Aug 2018)
  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)
  8 by 10 by 16 MPI processor grid
Created 1048576 atoms
  Time spent = 0.048476 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) = 2.699 | 2.703 | 2.708 Mbytes
Step Temp E_pair E_mol TotEng Press
       0         1.44   -6.7733681            0   -4.6133701   -5.0196704
 1000000   0.65695755   -5.7125359            0   -4.7271005   0.48799127
Loop time of 723.716 on 1280 procs for 1000000 steps with 1048576 atoms

Performance: 596918.946 tau/day, 1381.757 timesteps/s
99.5% CPU use with 1280 MPI tasks x 1 OpenMP threads

MPI task timing breakdown:
Section |  min time  |  avg time  |  max time  |%varavg| %total
---------------------------------------------------------------
Pair    | 424.38     | 435.47     | 461.05     |  26.2 | 60.17
Neigh   | 59.782     | 60.365     | 62.991     |   3.9 |  8.34
Comm    | 193.24     | 219.39     | 231.11     |  38.5 | 30.31
Output  | 0.00013494 | 0.00085223 | 0.0088639  |   0.0 |  0.00
Modify  | 6.4813     | 6.6462     | 7.541      |   5.6 |  0.92
Other   |            | 1.841      |            |       |  0.25

Nlocal:    819.2 ave 845 max 786 min
Histogram: 3 2 34 115 256 372 315 137 33 13
Nghost:    2417.97 ave 2468 max 2369 min
Histogram: 8 31 81 216 314 327 202 76 22 3
```

```
Neighs:     30698 ave 32432 max 28796 min
Histogram: 4 16 47 194 306 325 245 103 34 6

Total # of neighbors = 39293494
Ave neighs/atom = 37.4732
Neighbor list builds = 50000
Dangerous builds not checked
Total wall time: 0:12:03
```

OpenFOAM

openFOAM is a free, opensource software which covers most areas of Engineering and Science. It can be used to solve very interesting problems in fields ranging from Turbulence, Heat transfer, Acoustics, Electromagnetics, complex fluid flows including chemical reactions, solid mechanics and a lot more. Please follow the link https://www.openfoam.com to get more information.

Description of Inputs for openFOAM

Input file is taken from NASA website which does wing body simulation. The data can be copied from /home/apps/Data/OpenFOAM path on PARAM Sanganak

Grid size: 10 million
Solver: sonicFoam
Iterations: 4000
Decomposition of grid is done using Metis.

Script of OpenFOAM

```
#!/bin/sh
#SBATCH -N 50
#SBATCH --ntasks-per-node=40
#SBATCH --threads-per-core=1
#SBATCH --ntasks=2000
#SBATCH --time=06:50:20
#SBATCH --job-name=openfoam
#SBATCH --error=job.%J.err_16_node_40
#SBATCH --output=job.%J.out_16_node_40
#SBATCH --partition=standard
###SBATCH --nodelist=cn[175-190]
##SBATCH --nodelist=cn[013-028]

ulimit -s unlimited
ulimit -c 0

#module load intel/2018.0.1.163
#module load intel/2019.5.281
module unload gnu8/8.3.0
module load openmpi-3.1.0_gcc_4.8.5

source /home/shwetad/OpenFOAM/GCC-openmpi/openfoam_bashrc_gcc_openmpi
```

```
#source
/opt/ohpc/pub/intel2018/compilers_and_libraries_2018.1.163/linux/mkl/bin/mk
lvars.sh intel64
export I_MPI_FALLBACK=disable
export I_MPI_FABRICS=shm:dapl
export I_MPI_DEBUG=5
export I_MPI_PIN_PROCESSOR_LIST=0-39
###########MXM Optimization###########
export  I_MPI_DAPL_SCALABLE_PROGRESS=1
export I_MPI_RDMA_TRANSLATION_CACHE=1
export I_MPI_FAIR_CONN_SPIN_COUNT=2147483647
export I_MPI_FAIR_READ_SPIN_COUNT=2147483647
#export I_MPI_ADJUST_REDUCE 2, I_MPI_ADJUST_BCAST 0
export I_MPI_RDMA_TRANSLATION_CACHE=1
export I_MPI_RDMA_RNDV_BUF_ALIGN=65536
export I_MPI_SPIN_COUNT=121
export  I_MPI_DAPL_DIRECT_COPY_THRESHOLD=65536
#export I_MPI_DAPL_UD=enable
##############################################
source /home/shwetad/OpenFOAM/Intel-2018/openfoam_bashrc_2018
#source /home/shwetad/OpenFOAM/openfoam_bashrc
export OMP_NUM_THREADS=1
cd /home/shwetad/OpenFOAM_DATA/NSM
#export FI_PROVIDER=mlx/ofi/verbs

rm -rf processor*
decomposePar
(time mpirun -np 2000 sonicFoam -parallel) 2>&1 | tee out_4000_NSM_50Node
```

Output Values after 4000 iterations:

```
forceCoeffs forces write:
    Cm    = -8.50123
    Cd    = 0.0327941
    Cl    = -1.926
    Cl(f) = -9.46423
    Cl(r) = 7.53823
```

The said iterations complete in 2minutes 50 seconds on 50 nodes.

WRF Application

The **W**eather **R**esearch and **F**orecasting **(WRF)** Model is a next-generation mesocale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs. WRF is suitable for a broad spectrum of applications across scales ranging from meters to thousands of kilometers. WRF was developed at the National Center for Atmospheric Research (NCAR) which is operated by the University Corporation for Atmospheric Research (UCAR), USA.

More information about WRF may please be found at:
https://www.mmm.ucar.edu/weather-research-and-forecasting-model

For a reference run, the dataset used is as following with model simulation time being reduced to 15 minutes:

Dataset: Single domain, large size. 2.5 km CONUS, June 4, 2005

(Ref: https://www2.mmm.ucar.edu/wrf/WG2/benchv3/#_Toc212961289)

The WRF input files used for reference run are present in */home/apps/Data/WRF/input/run*

```
# Changes/Suggestions to namelist.input

&time_control
 run_hours                           = 0,
 run_minutes                         = 15,
 io_form_history                     = 11,        //parallel-netcdf
 io_form_restart                     = 11,
 io_form_input                       = 11,
 io_form_boundary                    = 11,            //serial-netcdf
 io_form_auxhist2                    = 2,

&dynamics
 use_baseparam_fr_nml                = .t.,


&namelist_quilt               // For no. of nodes (e.g. greater than 32
                              nodes) using quilt servers gives better
                              performance
 nio_tasks_per_group = 0,
 nio_groups = 1,
```

1. WRF job submission SLURM script

   The following reference job script is placed in */home/apps/Data/WRF/input/run/wrf_4n.sh*

```
#!/bin/bash

#SBATCH -N 4
#SBATCH --ntasks-per-node=40
#SBATCH --time=00:30:00
#SBATCH --job-name=WRF_CONUS
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out
#SBATCH --partition=standard
cd $SLURM_SUBMIT_DIR

###Loading WRF environment
module load wrf/3.8.1/intel2018

###Creating list of nodes to map WRF MPI processes
mpiexec.hydra -n $SLURM_NTASKS hostname > hosts.txt
sort -u hosts.txt > hosts_wrf.txt
sed -i 's/$/:20/' hosts_wrf.txt

###Two OpenMP threads per MPI rank
WRFMPI=` expr $SLURM_NTASKS / 2 `

###Setting Intel MPI environment
```

```
export I_MPI_DEBUG=9
export I_MPI_FALLBACK=disable

      (time mpiexec.hydra --machinefile hosts_wrf.txt -env I_MPI_PIN_DOMAIN
omp:compact -env OMP_NUM_THREADS=2 -env KMP_STACKSIZE=200m -n $WRFMPI
wrf.exe) >&  4n.2omp.wrf.out

### To save execution command to out file
echo "(time mpiexec.hydra --machinefile hosts_wrf.txt -env I_MPI_PIN_DOMAIN
omp:compact -env OMP_NUM_THREADS=2 -env KMP_STACKSIZE=200m -n $WRFMPI
wrf.exe)" >> 4n.2omp.wrf.out
```

2. WRF Output Snippet

```
$tail rsl.out.0000
Timing for main: time 2005-06-04_06:14:45 on domain   1:    2.16898 elapsed seconds
Timing for main: time 2005-06-04_06:15:00 on domain   1:    2.16480 elapsed seconds
wrf: SUCCESS COMPLETE WRF
```

The above workload on four compute nodes took approximate execution time of 3min15sec

NAMD Application

Nano Scale Molecular Dynamics (NAMD) software for molecular dynamics simulation is designed for high-performance simulations of Large Macro Molecular system on parallel computers. This software also makes use of GPGPUs.
More information about NAMD may please be found at
http://www.ks.uiuc.edu/Research/namd/

Citation:
"NAMD was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign."

James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26:1781-1802, 2005.

For a reference run, NAMD version 2.13 with following dataset has been used:
Dataset: STMV (virus) benchmark (1,066,628 atoms, periodic, PME)
(Ref: https://www.ks.uiuc.edu/Research/namd/utilities/)

Please follow the examples to get familiarized with writing scripts for using NAMD with CPU cores and GPGPUs.

The NAMD input files and SLURM job scripts used for reference runs are present in */home/apps/reference/namd/*

1. Contents of *stmv.namd* file used for a reference run with *numsteps* increased to 5000

```
#############################################################
## ADJUSTABLE PARAMETERS                                   ##
#############################################################
structure          stmv.psf
coordinates        stmv.pdb


#############################################################
## SIMULATION PARAMETERS                                   ##
#############################################################

# Input
paraTypeCharmm      on
parameters          par_all27_prot_na.inp
temperature         298


# Force-Field Parameters
exclude             scaled1-4
1-4scaling          1.0
cutoff              12.
switching           on
switchdist          10.
pairlistdist        13.5


# Integrator Parameters
timestep            1.0
nonbondedFreq       1
fullElectFrequency  4
stepspercycle       20


# Constant Temperature Control
langevin            on    ;# do langevin dynamics
langevinDamping     5     ;# damping coefficient (gamma) of 5/ps
langevinTemp        298
langevinHydrogen    off    ;# don't couple langevin bath to hydrogens


# Constant Pressure Control (variable volume)
useGroupPressure      yes ;# needed for rigidBonds
useFlexibleCell       no
useConstantArea       no

langevinPiston        on
langevinPistonTarget  1.01325 ;#  in bar -> 1 atm
langevinPistonPeriod  100.
langevinPistonDecay   50.
langevinPistonTemp    298

cellBasisVector1     216.832    0.   0.
cellBasisVector2     0.   216.832    0.
cellBasisVector3     0.    0   216.832
cellOrigin           0.    0.   0.

PME                  on
PMEGridSizeX         216
PMEGridSizeY         216
PMEGridSizeZ         216
```

```
# Output
outputName          stmv-output

outputEnergies      20
outputTiming        20

numsteps            5000
```

2. NAMD job submission SLURM script for GPGPUs
   The reference job script is placed at */home/apps/reference/namd/namd_gpu_2.13.sh*

```
#!/bin/bash

#SBATCH -N 5
#SBATCH --ntasks-per-node=40
#SBATCH --time=00:10:00
#SBATCH --job-name=NAMD_GPU
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out
#SBATCH --partition=gpu

cd $SLURM_SUBMIT_DIR
module load namd/2.13/impi2019v5/cuda
module unload gnu8/8.3.0
module load intel/2019.5.281

mpiexec.hydra -n $SLURM_NTASKS hostname > hosts.txt
sort -u hosts.txt > hosts_namd.txt
sed -i 's/$/:2/' hosts_namd.txt

NAMDMPI=` expr $SLURM_NTASKS / 20 `
export I_MPI_DEBUG=9
#export I_MPI_FABRICS=shm:dapl
export FI_PROVIDER=mlx
export I_MPI_FALLBACK=0

      (time mpiexec.hydra --machinefile hosts_namd.txt -n $NAMDMPI namd2
+ppn 19 +pemap 1-19,21-39 +commap 0,20 +setcpuaffinity +isomalloc_sync
+idlepoll ./stmv.namd  +devices 0,1)  >& 5N.namd.cuda.out
echo "(time mpiexec.hydra --machinefile hosts_namd.txt -n $NAMDMPI namd2
+ppn 19 +pemap 1-19,21-39 +commap 0,20 +setcpuaffinity +isomalloc_sync
+idlepoll ./stmv.namd  +devices 0,1)" >> 5N.namd.cuda.out
mpiicc -v       >> 5N.namd.cuda.out
icc -v          >> 5N.namd.cuda.out
which namd2     >> 5N.namd.cuda.out
```

3. NAMD job submission SLURM script for CPUs
   The reference job script is placed at */home/apps/reference/namd/namd_cpu_2.13.sh*

```
#!/bin/bash

#SBATCH -N 16
#SBATCH --ntasks-per-node=40
#SBATCH --time=00:15:00
#SBATCH --job-name=NAMD
```

```
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out
#SBATCH --partition=standard

cd $SLURM_SUBMIT_DIR
module load namd/2.13/impi2019v5/cpu
module unload gnu8/8.3.0
module load intel/2019.5.281

mpiexec.hydra -n $SLURM_NTASKS hostname > hosts.txt
sort -u hosts.txt > hosts_namd.txt
sed -i 's/$/:2/' hosts_namd.txt

NAMDMPI=` expr $SLURM_NTASKS / 20 `
export I_MPI_DEBUG=9
#export I_MPI_FABRICS=shm:dapl
export FI_PROVIDER=mlx
export I_MPI_FALLBACK=0
mpiicc -v

      (time mpiexec.hydra --machinefile hosts_namd.txt -n $NAMDMPI namd2
+ppn 19 +pemap 1-19,21-39 +commap 0,20 +setcpuaffinity +isomalloc_sync
+idlepoll ./stmv.namd)  >&  16N.namd.cpu.out
echo "(time mpiexec.hydra --machinefile hosts_namd.txt -n $NAMDMPI namd2
+ppn 19 +pemap 1-19,21-39 +commap 0,20 +setcpuaffinity +isomalloc_sync
+idlepoll ./stmv.namd)" >>  16N.namd.cpu.out
mpiicc -v       >> 16N.namd.cpu.out
icc -v          >> 16N.namd.cpu.out
which namd2     >> 16N.namd.cpu.out
```

4. NAMD Output Snippet

```
TIMING: 5000  CPU: 53.5804, 0.010235/step  Wall: 53.6822, 0.0102851/step, 0 hours remaining,
5024.433594 MB of memory in use.
ETITLE:      TS           BOND          ANGLE          DIHED          IMPRP
ELECT             VDW       BOUNDARY           MISC        KINETIC               TOTAL
TEMP      POTENTIAL         TOTAL3        TEMPAVG            PRESSURE      GPRESSURE
VOLUME       PRESSAVG      GPRESSAVG

ENERGY:    5000    368697.5464    279967.9454     81941.7526      5087.5336       -
4524058.5568    385721.1155         0.0000         0.0000    945098.3696       -2457544.2937
297.2577  -3402642.6633  -2449021.7220       297.4163           -83.0695       -13.9811
10199588.8034        13.0980        11.6157
```

5. The above workload took approximate execution times as:

S. No.	Resource Type	No. of Compute Nodes	ns/day	Execution Time (sec)
1	GPGPU (with CPU)	5	8	75
2	CPU	16	8	75

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:

The support and the resources provided by PARAM Sanganak under the National Supercomputing Mission, Government of India at the Indian Institute of Technology, Kanpur are gratefully acknowledged.

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

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

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

Getting Help – PARAM Sanganak Support

We suggest that you kindly visit the link https://paramsanganak.iitk.ac.in/faq to know if the problem you are facing has already been reported by some other user and has already been solved. If this is the case, you will find a summary of the solution provided by the system administrator there. If you do not find any mention of the issue being faced by you, please refer to these four easy steps to generate a Ticket related to the issue you are experiencing.

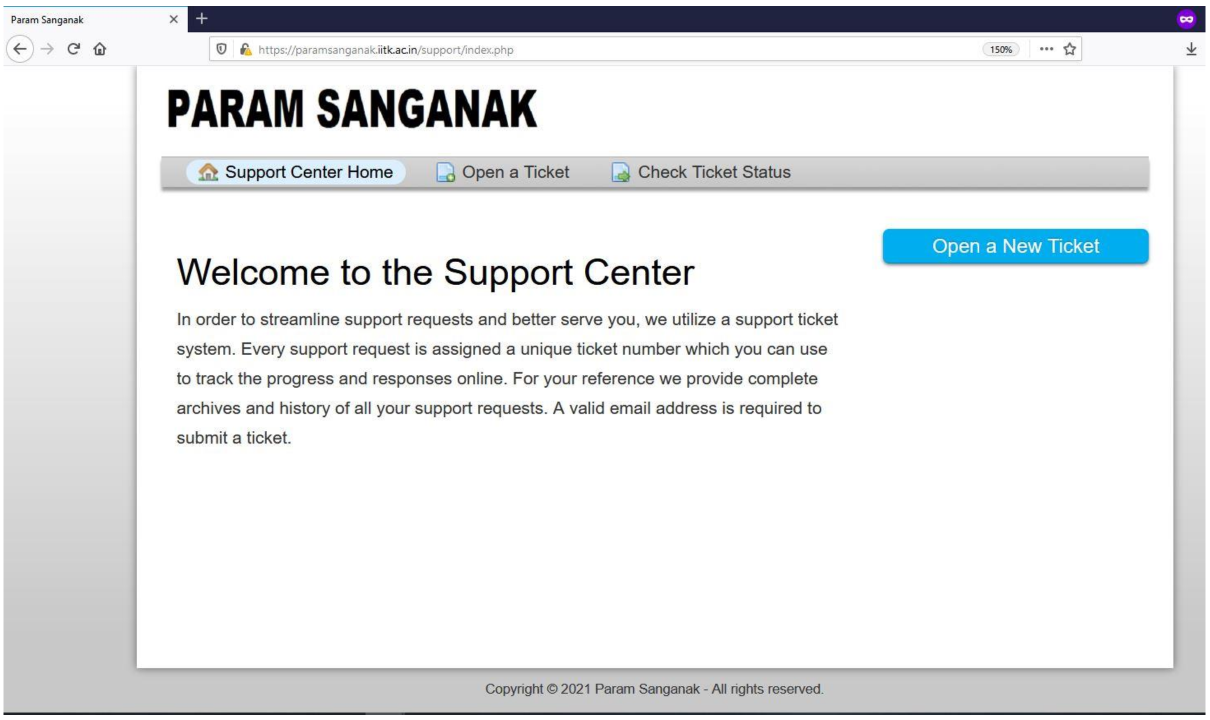
Your Ticket will be assisted by the Sanganak Support team. The ticket generated will be closed only when the related issue gets resolved.

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

➢ Steps to Create a new ticket:

1) Place the URL (http://paramsanganak.iitk.ac.in/support) in your browser.

2) Click "Open a New Ticket". Refer to Figure: 36 for the same.



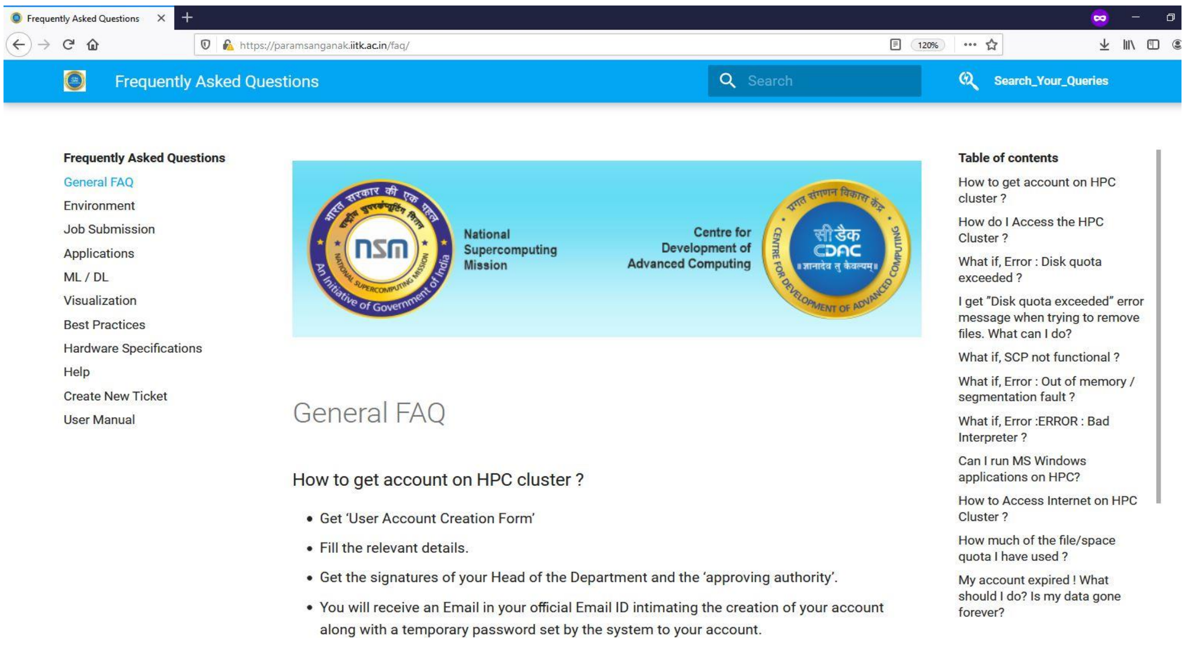
Welcome to the Support Center

In order to streamline support requests and better serve you, we utilize a support ticket system. Every support request is assigned a unique ticket number which you can use to track the progress and responses online. For your reference we provide complete archives and history of all your support requests. A valid email address is required to submit a ticket.

Copyright © 2021 Param Sanganak - All rights reserved.

Figure 36 – PARAM Sanganak Dashboard

3) Search your issue in the FAQ first, before raising a ticket. Still if your issue is not resolved then user may raise a ticket by clicking "Create New Ticket". Refer to Figure: 37 for the same.



Frequently Asked Questions
https://paramsanganak.iitk.ac.in/faq/
Search
Search_Your_Queries
Frequently Asked Questions
General FAQ
Environment
Job Submission
Applications
ML / DL
Visualization
Best Practices
Hardware Specifications
Help
Create New Ticket
User Manual
National Supercomputing Mission
Centre for Development of Advanced Computing
Table of contents
How to get account on HPC cluster ?
How do I Access the HPC Cluster ?
What if, Error : Disk quota exceeded ?
I get "Disk quota exceeded" error message when trying to remove files. What can I do?
What if, SCP not functional ?
What if, Error : Out of memory / segmentation fault ?
What if, Error :ERROR : Bad Interpreter ?
Can I run MS Windows applications on HPC?
How to Access Internet on HPC Cluster ?
How much of the file/space quota I have used ?
My account expired ! What should I do? Is my data gone forever?

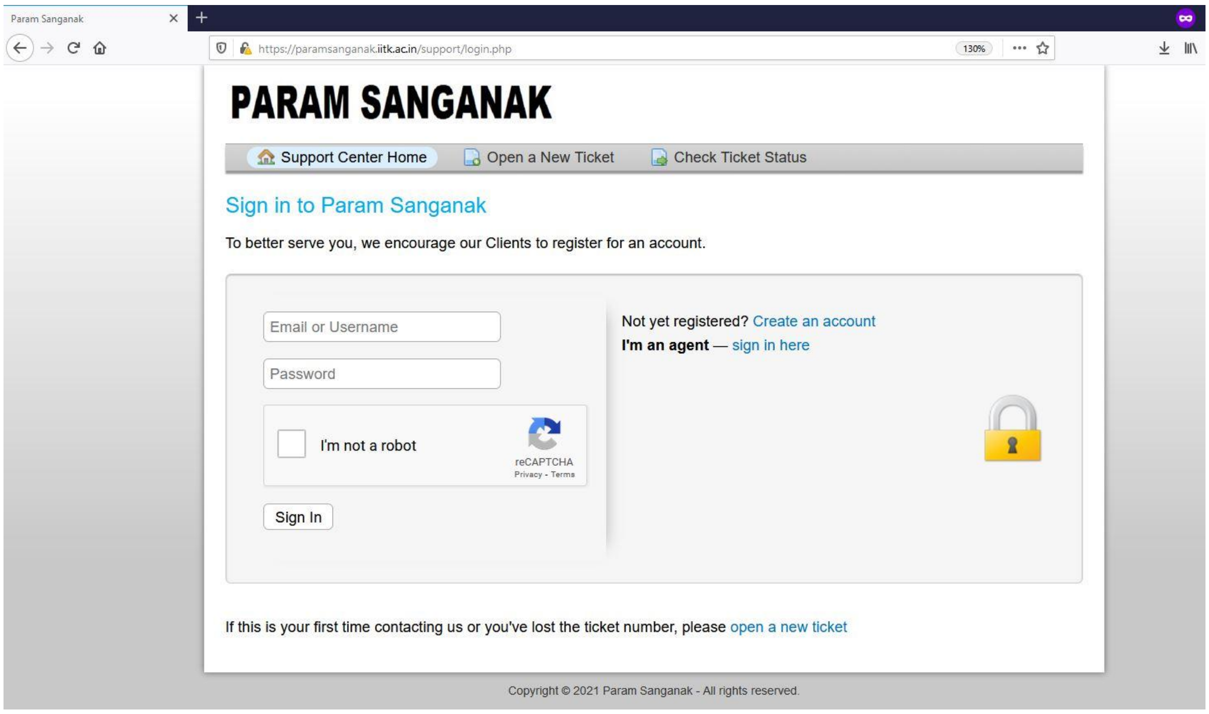
General FAQ

How to get account on HPC cluster ?

- Get 'User Account Creation Form'
- Fill the relevant details.
- Get the signatures of your Head of the Department and the 'approving authority'.
- You will receive an Email in your official Email ID intimating the creation of your account along with a temporary password set by the system to your account.

Figure 37 – FAQ

4) Sign in by providing the Username and Password. Refer to Figure: 38.



Sign in to Param Sanganak

To better serve you, we encourage our Clients to register for an account.

Not yet registered? Create an account

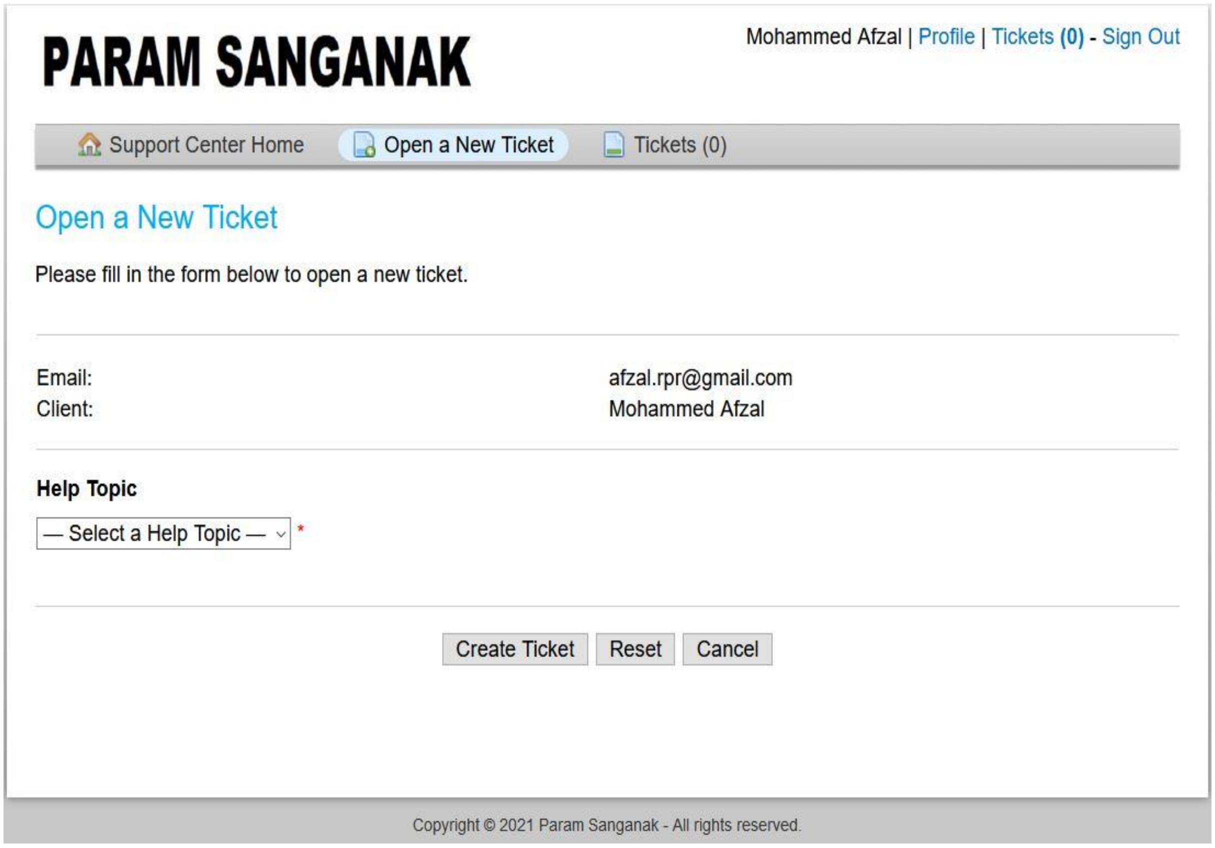
I'm an agent — sign in here

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

Copyright © 2021 Param Sanganak - All rights reserved.

Figure 38 - Signing page.

5) Select a "Help Topic" From the Dropdown and then Click on "Create Ticket". Refer to Figure: 39 for the same.



Open a New Ticket

Please fill in the form below to open a new ticket.

Email:	afzal.rpr@gmail.com
Client:	Mohammed Afzal

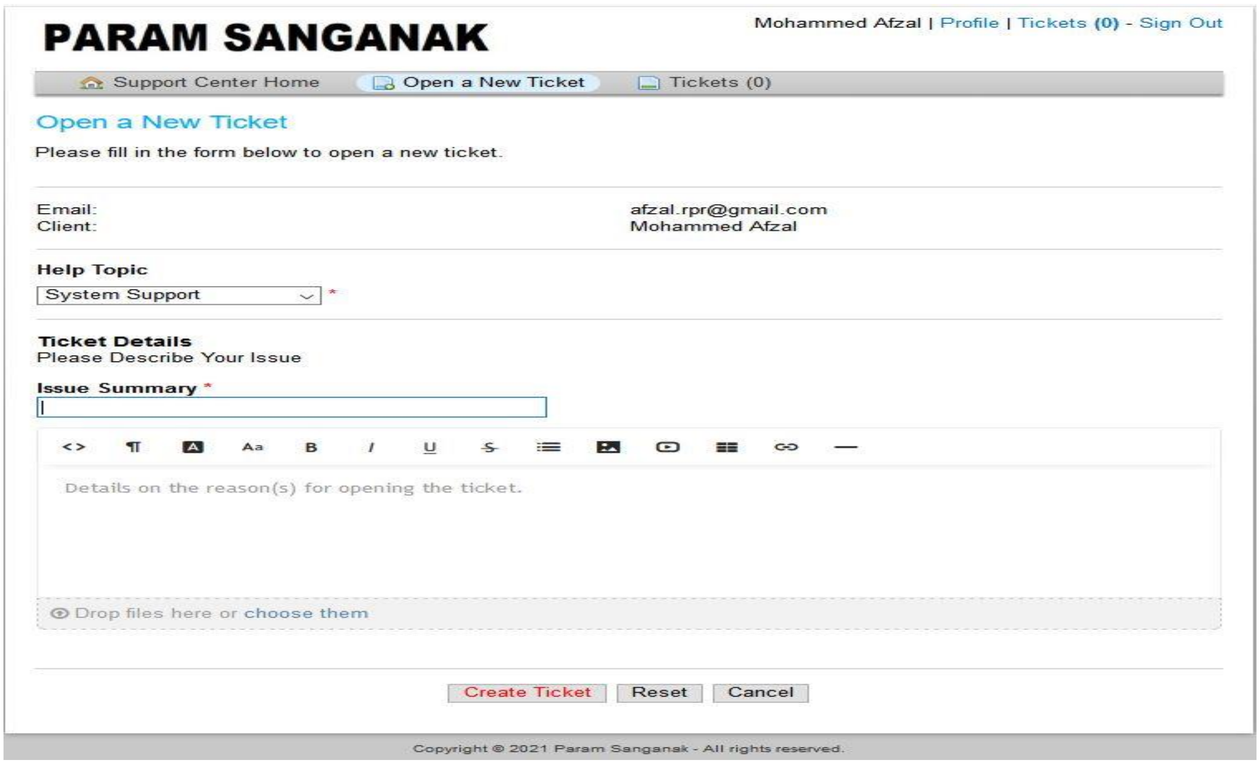
Help Topic

— Select a Help Topic — *

Copyright © 2021 Param Sanganak - All rights reserved.

Figure 39 – Raise a Ticket.

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



Open a New Ticket

Please fill in the form below to open a new ticket.

Email: afzal.rpr@gmail.com
Client: Mohammed Afzal

Help Topic
System Support *

Ticket Details
Please Describe Your Issue

Issue Summary *

Details on the reason(s) for opening the ticket.

Copyright © 2021 Param Sanganak - All rights reserved.

Figure 40 - Snapshot of ticket generation

Once the Ticket will be generated, an acknowledgement e-mail will be sent to your official e-mail address/email provided at the time of user creation. The e-mail will also contain the Ticket number with reference to the ticket that you have generated.

In case of any difficulty while accessing Sanganak-Support you can reach us via e-mail at sanganaksupport@iitk.ac.in

References

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

End of the Document