PARAM Ganga

Quick Start Guide

Ver. 1.0

Last updated: February 28, 2022

www.cdac.in

First Things First

This document is the quick start guide for the PARAM Ganga Supercomputing facility at IIT Roorkee. It covers brief details about the hardware infrastructure, compilers and modules, submitting jobs, retrieving the results on to user's Laptop/ Desktop etc.

The supercomputer PARAM Ganga is based on a heterogeneous and hybrid configuration of Intel Xeon Cascade lake processors, and NVIDIA Tesla V100.

The total peak computing capacity of 1.67 (CPU+GPU+HM) PFLOPS performance.

First login

Whenever the newly created user on PARAM Ganga tries to login with the user Id and password (temporary, system generated) provided over the Email through PARAM Ganga support, he/she will next be prompted to create a "new password" of their choice which will change the temporary, system generated password. This will enable you to keep your account secure. It is recommended that you have a strong password 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.

System Access

Accessing the cluster

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

- You may access login node through ssh.
- The login node is primary gateway to the rest of the cluster, which has a job scheduler (called Slurm). You may submit jobs to the queue and they will run when the required resources are available.
- Please do not run programs directly on login node. Login node is use to submit jobs, transfer data and to compile source code. (If your compilation takes more than a few minutes, you should submit the compilation job into the queue to be run on the cluster.)

Remote Access

Using SSH in Windows

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

(e.g. [username]@paramganga.iitr.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 Ganga Login Node, with the username

```
user1: ssh paramganga.iitr.ac.in
```

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

If you are accessing the facility outside IITR network, please use port **4422**

```
ssh [username]@[hostname] –p 4422
```

Password

How to change the user password?

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

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

Tools to accessing cluster

MobaXterm (Windows installable application):

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

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

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 Ganga system through your local systems (laptop/desktop).

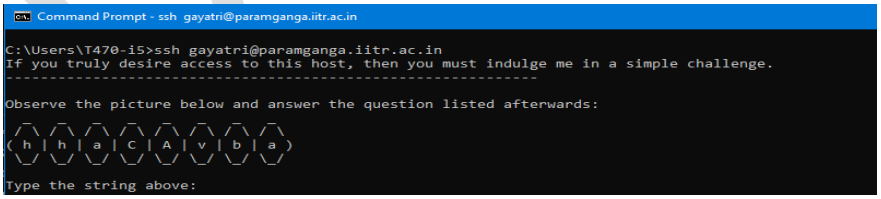

Figure 4 - A snapshot of "scp" command using Windows command prompt.

PowerShell (Windows native application):

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

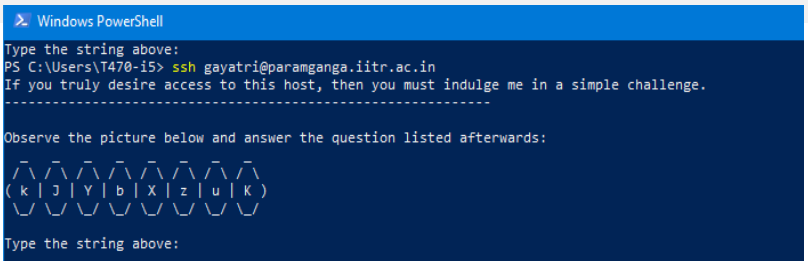

Figure 5 - A snapshot of "scp" command using Windows PowerShell.

Introduction

System Hardware Specifications

PARAM Ganga 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 nodes: 214
  - GPU nodes: 20
  - High Memory nodes:78

CPU Compute Nodes: 214

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

CPU Nodes: 214	
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 = 10,272 cores Total Memory=41088 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 Memory= 768 GB, DDR4 2933 MHz SSD = 480 GB (local scratch) per node	 Total Cores = 3744 cores Total Memory=59904 GB

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 Cores = 40, 2.5 GHz Memory= 192 GB, DDR4 2933 MHz SSD = 480 GB (local scratch) per node **2*NVidia V100 per node** GPU Cores per node= 2*5120= 10240 GPU Memory = 16 GB HBM2 per NVidia V100	Total Cores = 800 cores Total Memory= 3840 GB

Operating System

Operating system on PARAM Ganga is Linux – CentOS 7.9

Commented [T1]:

Commented [T3R2]:

Commented [T2]: Need to change architecture diagram

Software Stack

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 1.

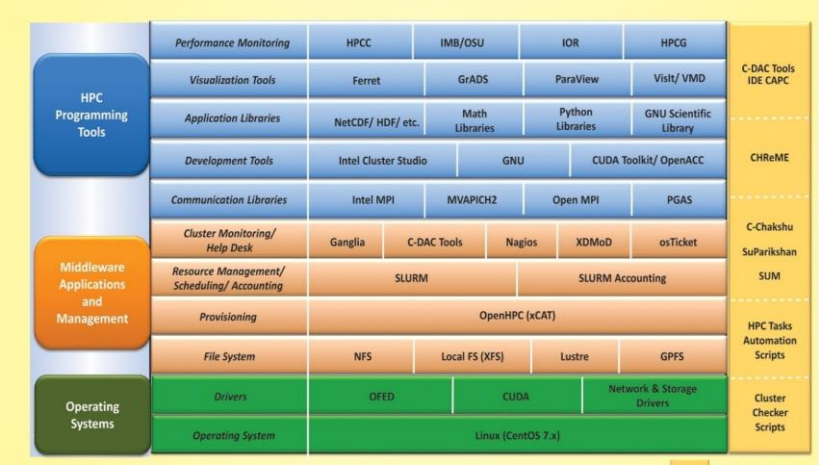

Figure 1 – Software Stack

Modules

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

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

```
$ module list
```

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

```
$ module avail
```

```
---------------------------------------------------- /opt/ohpc/admin/modulefiles ----------------------
   spack/0.12.1

------------------------------------------------------- /opt/ohpc/pub/modulefiles ------------------------
   DL-CondaPy/3.7              hwloc/2.1.0                    oneapi/init_opencl/2022.0.2
   DL-IntelPy/3.7              llvm5/5.0.1                    oneapi/inspector/2022.0.0
   EasyBuild/3.9.4             oneapi/advisor/2022.0.0        oneapi/intel_ipp_ia32/2021.5.2
   Rapids/22.02                oneapi/ccl/2021.5.1            oneapi/intel_ipp_intel64/2021.5.2
   autotools                   oneapi/clck/2021.5.0           oneapi/intel_ippcp_ia32/2021.5.1
   charliecloud/0.11           oneapi/compiler-rt/2022.0.2    oneapi/intel_ippcp_intel64/2021.5.1
   clustershell/1.8.2          oneapi/compiler-rt32/2022.0.2  oneapi/itac/2021.5.0
   cmake/3.15.4                oneapi/compiler/2022.0.2       oneapi/mkl/2022.0.2
   compiler/intel/2018_4       oneapi/compiler32/2022.0.2     oneapi/mkl32/2022.0.2
   compiler/intel/2019_5       oneapi/dal/2021.5.3            oneapi/mpi/2021.5.1
   compiler/intel/2020_4 (D)   oneapi/debugger/2021.5.0       oneapi/oclfpga/2022.0.2
   cuda/8.0                    oneapi/dev-utilities/2021.5.2  oneapi/tbb/2021.5.1
   cuda/9.0                    oneapi/dnnl-cpu-gomp/2022.0.2  oneapi/tbb32/2021.5.1
   cuda/9.2                    oneapi/dnnl-cpu-iomp/2022.0.2  oneapi/vpl/2022.0.0
   cuda/10.0                   oneapi/dnnl-cpu-tbb/2022.0.2   oneapi/vtune/2022.0.0
   cuda/10.1                   oneapi/dnnl/2022.0.2           openmpi/4.0.5
   cuda/10.2                   oneapi/dpct/2022.0.0           papi/5.7.0
   cuda/11.0                   oneapi/dpl/2021.6.0            prun/1.3
   cuda/11.6             (D)   oneapi/icc/2022.0.2            singularity/3.4.1
   gnu8/8.3.0                  oneapi/icc32/2022.0.2          valgrind/3.15.0

  Where:
   D:  Default Module
```

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

```
$ module load DL-CondaPy/3.7
```

Running Interactive Jobs

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

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

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

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

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

Where xxxxx is the job id.

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

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

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

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

Managing Jobs through its Lifecycle

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

```
module                                  # This command lists all the available
module load compiler/intel/2018.4       # This will load the intel compilers into
your
environment
module unload compiler/intel/2018.4     # This will remove all environment
```

A simple Slurm job script

```
#!/bin/sh
#SBATCH -N                          // specifies number of
#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
```

List Partition

sinfo displays information about nodes and partitions(queues).
$ sinfo

```
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
queue_test   up   1:00:00       312     idle cn[001-214],hm[001-078],gpu[001-020]
```

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 queue_test
#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 queue_test simple  user1   R   0:31  1    atom01
```

3. 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"

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

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.

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:

a. Compilations are done on the login node. Only the execution is scheduled via SLURM on the compute/GPU nodes

b. Upon Submission of a Job script, each job gets a unique Job Id. This can be obtained from the '**squeue**' command.

c. 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@iitr.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 --partition=standard / Partition or queue name

// To load the module //
module load

compiler/intel/2018.2.199 cd <Path

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>
```

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_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]

Standard Application Programs on PARAM Ganga

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

LAMMPS Applications

LAMMPS is an acronym for **L**arge-scale **A**tomic/ **M**olecular **M**assively **P**arallel **S**imulator. This is extensively used in the fields of Material Science, Physics, Chemistry and may others.

More information about LAMMPS may please be found at https://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
```

```
#!/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
```

2. Sample SLURM script for LAMMPS for 32 nodes

```
#!/bin/sh

#SBATCH -N 32
#SBATCH --ntasks-per-node=48
##SBATCH --time=52:50:20
#SBATCH --job-name=lammps
#SBATCH --error=job.%J.err_32_node_40
#SBATCH --output=job.%J.out_32_node_40
#SBATCH --partition=queue_test
module load compiler/intel/2018_4

export
 LD_LIBRARY_PATH=/opt/ohpc/pub/compiler/intel/2018_4/compilers_and_libraries_2018.5.274/
 linux/mkl/lib/intel64:$LD_LIBRARY_PATH
export
 MKL_ROOT=/opt/ohpc/pub/compiler/intel/2018_4/compilers_and_libraries_2018.5.274/linux/m
 kl:$MKL_ROOT

source
 /opt/ohpc/pub/compiler/intel/2018_4/compilers_and_libraries_2018.5.274/linux/tbb/bin/tb
 bvars.sh intel64

module unload gnu8/8.3.0

export I_MPI_FALLBACK=disable
export I_MPI_FABRICS=shm:dapl
export I_MPI_DEBUG=5

cd /home/manjunath/LAMMPS/lammps-29Sep2021/bench
```

```
ulimit -s unlimited # stack
ulimit -d unlimited # data area
ulimit -c unlimited
ulimit -a
ulimit -n

export OMP_NUM_THREADS=1

time mpiexec.hydra -n $SLURM_NTASKS -genv OMP_NUM_THREADS 1
 /home/<username>/LAMMPS/lammps-29Sep2021/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


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

      ---------------------------------------------------------------------------
      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
```

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 Ganga

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=queue_test
###SBATCH --nodelist=cn[175-
190] ##SBATCH --
nodelist=cn[013-028]

ulimit -s
unlimited ulimit
-c 0

#module load
intel/2018.0.1.163 #module
```

```
#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

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

&dynamics
 use_baseparam_fr_nml                = .t.,


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

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=queue_test
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
```

DL Application

The DL Frame Work available in DL-Conda

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

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

	Name	Support	Version
DL Framework	Tensorflow	CPU and GPU	2.0.0
	Pytorch	CPU and GPU	1.3.1
	Theano	CPU and GPU	1.0.5
	Caffe	CPU and GPU	1.0
	keras	CPU and GPU	2.3.1
Data science framework	RAPIDS	GPU	22.02
	pandas	CPU	1.3.5
	cupy	GPU	9.6.0

```
#!/bin/bash

#SBATCH -N 1                                    # Specify number of the nodes
#SBATCH -n 2                                    # Specify number of the tasks
to run
#SBATCH -p gpu                                  # Specify GPU Partition name
#SBATCH --gres=gpu:2                            # Specify number of the gpus
#SBATCH --output=dljob_%J.out                   # Specify output file to store
output of the model
#SBATCH --error=dljob_%J.err                    # Specify error file
#SBATCH --time=03:00:00                         # Specify time limit {HH:MM:SS}

# Load conda-python module using module load <module>
module load DL-CondaPy/3.7       #THIS MODULE HAS ALL POPULAR DL PACKAGES
##python <yourcode.py>
python yourcode.py
```