Computer Science
California State
Polytechnic
University, Pomona

3801 West Temple
Avenue
Pomona, CA 91768

Cal Poly HPC Environment

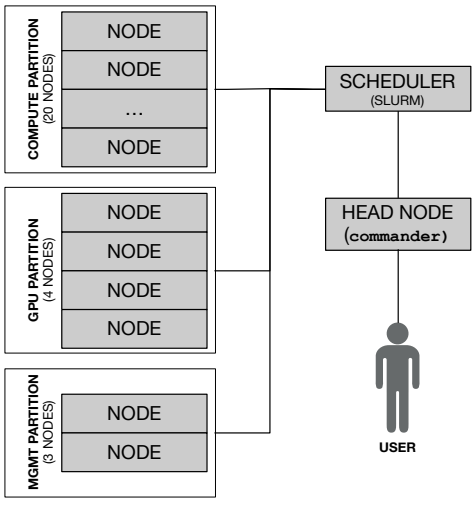
Week 03

Computer Science
California State
Polytechnic
University, Pomona

3801 West Temple
Avenue
Pomona, CA 91768

Overview

- State-of-the-art research and teaching environment for faculty and students
- Consists of many computers, referred to as nodes, connected together via network and managed by a special software
- Based upon the HP ProLiant server platform
  - 2 DL360 Management nodes
  - 20 DL160 Compute nodes
  - 4 GPU nodes with total of 8 Tesla P100 GPUs
  - Total 3.3TB of RAM connected through a dedicated internal 40GBit InfiniBand switching fabric
  - 10GBit external ethernet connections
- Overall throughput is currently 36.6 Tflp (i.e. 36.6×10^{12} floating point operations per second) in double precision mode or 149.6 Tflp in half precision mode
- Uses SLURM for workload manager
- Supports C/C++, Python, Julia, Java, and R programming languages
- This course will mainly use Python and C/C++ for assignments



THUANG © 2024. All rights reserved.
CS 3700 – Parallel Processing

Access to HPC

- All assignments will be done using ZFS and our HPC environments. General UNIX experience and text editor are necessary. See **General Computing Environment Guide**
  - **HPC (hpc.cpp.edu)** for MPI and GPU assignments
  - **ZFS (login,cpp,edu)** for Thread assignments
- The HPC is accessible from any computer in the CS Lab (within the Cal Poly network)
- **VPN** is required when access from outside of Cal Poly
- **SSH** is required for all remote terminal access
- **SCP** is required for all file transfer (e.g., downloading your completed assignment zip file to your local computer in order to upload to Canvas)
- The **`hpc.cpp.edu`** is the head node (i.e. **`commander`**). This is the host to submit parallel processing jobs

```
$ ssh thuang@hpc.cpp.edu
Unauthorized use of Cal Poly Pomona computer and networking resources is
prohibited. If you log on to this computer system, you acknowledge your
awareness of and concurrence with the Cal Poly Pomona Acceptable Use Policy.
The University will prosecute violators to the full extent of the law.

thuang@hpc.cpp.edu's password:
Last login: Sat Jan 20 11:47:00 2024 from 10.104.192.26

thuang@commander ~ $ cd dev

thuang@commander ~/dev $ ls -l split.*
-rwxr-xr-x 1 thuang cpp 451 Sep  8 10:41 split.py
-rwxr-xr-x 1 thuang cpp 317 Sep  8 09:08 split.sh
```


Slurm

- Running programs on an HPC environment requires learning new tools. Remember, we are running jobs on a computing cluster, not a single computer
- We use Slurm for submitting and monitoring jobs on the HPC
- Slurm is an open-source cluster management and job management system
- Typical Slurm commands
  - **`srun`**: obtain a job allocation and execute an application. Common parameters
    - **`-p`**: partition (**`compute`** | **`gpu`**). There are two types of partitions in the CPP HPC system
      - **compute** for CPU-based computing
      - **gpu** for GPU-based computing
    - **`-N`**: nodes requireed for the job
    - **`-n`**: number of tasks to be launched
    - **`--pty`**: execute task zero in pseudo terminal mode
    - **`--gres`**: a comma delimited list of generic consumable resources (e.g. for GPU **`--gres=gpu:gp100gl:2`**).
  - **`sbatch`**: submit a batch script for later execution
  - **`sacct`**: display accounting data. Use **`-j <jobid>`** to see status of a specific job
  - **`sinfo`**: view status of the cluster's nodes and partitions
  - **`squeue`**: display information of jobs in queue

Computer Science
California State
Polytechnic
University, Pomona

3801 West Temple
Avenue
Pomona, CA 91768

Example MPI Program

```
#!/usr/bin/python3

# split.py - split process into 2 communicators
from mpi4py import MPI

world = MPI.COMM_WORLD
numprocs = world.Get_size() # get the current
myid = world.Get_rank()
color = myid%2
new_comm = world.Split(color)
new_id = new_comm.Get_rank()
new_nodes = new_comm.Get_size()

broad_val = color
new_comm.bcast(broad_val, root=0)
print("Old_proc['%d'] has new rank %d of %d received value %d"
   %(myid, new_id, new_nodes, broad_val))
```


Computer Science
California State
Polytechnic
University, Pomona

3801 West Temple
Avenue
Pomona, CA 91768

Using SBATCH

```
#!/bin/bash
#SBATCH --job-name=SPLIT
#SBATCH --output=SPLIT.txt
#SBATCH --mem-per-cpu=1024
#SBATCH --partition=compute
#SBATCH --nodes=10

. /etc/profile.d/modules.sh

module load openmpi/2.1.2
module load python/3/mpi4py/3.0.0

export MPI_HOME=/opt/openmpi-2.1.2
export PATH=${MPI_HOME}/bin:${PATH}
mpirun python3.4 split.py
```

`split.sh` script for sbatch submission

- Use SBATCH to configure jobs for the HPC
- The `#SBATCH` directive is to configure how the job should be run
  - **`--job-name`**: user-defined name for the job
  - **`--output`**: user-define execution log file name
  - **`--mem-per-cpu`**: required memory for each CPU
  - **`--partition`**: compute or gpu
  - **`--node`**: number of computing nodes
  - **`--gres`**: generic consumable resources (e.g. for GPU **`gpu:gp100gl:2`**)
- The reset of the file consists of execution sequence, like shell scripting

Computer Science
California State
Polytechnic
University, Pomona

3801 West Temple
Avenue
Pomona, CA 91768

Job Submission

```
thuang@commander ~/dev $ chmod ugo+x split.py

thuang@commander ~/dev $ sbatch split.sh
Submitted batch job 1072

thuang@commander ~/dev $ sacct -j 1072
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
1072             SPLIT    compute        cpp         10  COMPLETED      0:0
1072.batch       batch                   cpp          1  COMPLETED      0:0
1072.0           orted                   cpp          9  COMPLETED      0:0
```

```
thuang@commander ~/dev $ cat SPLIT.txt
Old_proc['0'] has new rank 0 of 5 received value 0
Old_proc['5'] has new rank 2 of 5 received value 1
Old_proc['3'] has new rank 1 of 5 received value 1
Old_proc['6'] has new rank 3 of 5 received value 0
Old_proc['7'] has new rank 3 of 5 received value 1
Old_proc['4'] has new rank 2 of 5 received value 0
Old_proc['9'] has new rank 4 of 5 received value 1
Old_proc['8'] has new rank 4 of 5 received value 0
Old_proc['2'] has new rank 1 of 5 received value 0
Old_proc['1'] has new rank 0 of 5 received value 1
```