Running Python on the Cluster

Getting Started Without MPI

The Basics

1. Load Python using *module load Python/3.9.5-GCCcore-10.3.0*
2. Check version using *python –version*
3. Ensure the Python shell opens by entering *python* then testing *print(‘hello world’)* within the shell. When confirmed, exit the shell with *ctrl d* or *exit()*
4. Now nano a new file with the .py extension and put the hello world function in it and save
5. Now create a simple job file with the .sh extension. Remember to change the ‘CHANGEME’ sections to match the name of your Python file

#!/bin/bash

# Template for running python script serially without MPI

#SBATCH --job-name=CHANGEME

#SBATCH -o CHANGEME.%j.out

#SBATCH -e CHANGEME.%j.err

#SBATCH -N 1

#SBATCH -n 1

# Load Python version and other modules

module load Python/3.9.5-GCCcore-10.3.0

# Runs Python script on current node only regardless of n

# python CHANGEME.py

# Runs Python Serially without MPI (multiple programs in parallel)

srun python CHANGEME.py

1. Now sbatch job.sh and see if you get an output file that says hello world. If you see “Batch script contains DOS line breaks” then go back to the console and enter the command *dos2unix filename.sh*
2. Now change the job.sh file so that -n 1 is -n 10 and sbatch again. You should get hello world 10 times. This confirms that your code is running parallel on multiple nodes

Getting Started With MPI

Check For Correct Modules Using *module spider*:

1. Python/3.9.5-GCCcore-10.3.0
2. mpi4py/3.1.4-gompi-2023a
3. openmpi4/4.1.1 (use command *swap mpich openmpi4/4.1.1* if needed)

Create A New Job File Called job.mpi

#!/bin/bash

# Template for running python script

# nodelist=c[010-017] #chasis 1

# nodelist=c[018-025] #chasis 2

# nodelist=c[026-033] #chasis 3

# nodelist=c[034-041] #chasis 4

# nodelist=c[042-049] #chasis 5

# nodelist=c[050-057] #chasis 6

# nodelist=c[058-065] #chasis 7

# nodelist=c[066-073] #chasis 8

# nodelist=c[074-081] #chasis 9

# nodelist=c[082-089] #chasis 10

# nodelist=c[090-097] #chasis 11

#SBATCH --job-name= CHANGEME

#SBATCH -o CHANGEME.%j.out # Places output files in sub-directory called Report

#SBATCH -e CHANGEME.%j.err # Places error files in sub-directory called Report

# Delete the two lines above for both files to stay in current directory

#SBATCH -N 1 # Number of servers to be used 88

#SBATCH -n 10 # Total number of cores to be used, there are 16 cores per node.

#SBATCH --time=100-00:30 # time (DD-HH:MM) after which job will be canceled

#SBATCH --partition=normal

# If you wish to specify a set of nodes, remove one comment symbol from the next line

##SBATCH --nodelist=c[010-025]

#SBATCH --mail-type=BEGIN,END # when to email user about job status

#SBATCH --mail-user= # email to send to user about job status updates

# Load Python, MPI, and other modules

module load Python/3.9.5-GCCcore-10.3.0

module load mpi4py/3.1.4-gompi-2023a

module load openmpi4/4.1.1

##module swap mpich openmpi4/4.1.1

# Enter the above line if the error file says a bunch of annoying stuff

#moduel load Python-bundle-PyPI/2023.06-GCCcore-12.3.0

# Runs Python script on current node

# python CHANGEME.py

# Runs Python Serially without MPI (multiple applications in parallel)

# srun python CHANGEME.py

# Runs Python with MPI (one application running parallel)

prun python CHANGEME.py

Now create a new py program called sendrecv.py

# Import the MPI module from mpi4py

from mpi4py import MPI

import time

# Initialize the MPI communicator

comm = MPI.COMM\_WORLD

# Get the rank, size, and processor name of the current process

rank = comm.rank

size = comm.size

name = MPI.Get\_processor\_name()

# Calculate a shared value based on the rank

shared = (rank + 1) \* 5

# Check each rank

if rank == 0:

# If rank is 0, send data to processes with ranks 1 and 2

data = shared

comm.send(data, dest=1)

comm.send(data, dest=2)

print('From rank', rank, 'we sent', data)

# Sleep for 5 seconds to allow other processes to complete

time.sleep(5)

elif rank == 1:

# If rank is 1, receive data from rank 0

data = comm.recv(source=0)

print('On rank', rank, 'we received:', data)

elif rank == 2:

# If rank is 2, receive data from rank 0

data = comm.recv(source=0)

print('On rank', rank, 'we received:', data)

Rename the “CHANGEME” sections on job.mpi to the name of your python file “sendrecv” and change -n to 3 since this program only uses 3 cores/ranks

Now we are ready to submit the MPI script

1. Save all your changes and make sure your files are still using unix format
2. In the console, enter *sbatch job.mpi*
3. If your error file is full, use *module swap mpich openmpi4/4.1.1* in the console again
4. Your output should come out to this:

[prun] Master compute host = c010

[prun] Resource manager = slurm

[prun] Launch cmd = mpirun python sendrecv.py (family=openmpi4)

From rank 0 we sent 5

On rank 1 we received: 5

On rank 2 we received: 5

Congrats, you now have Python and MPI at your disposal

Some notes:

* To enter inputs via command line or in your job file, import sys and use sys.argv to store inputs in. For example:

# Python program to demonstrate

# sys.argv

import sys

add = 0.0

# Getting the length of command

# line arguments

n = len(sys.argv)

for i in range(1, n):

add += float(sys.argv[i])

print ("the sum is :", add)

* Now during runtime you can add your inputs after python code.py [add inputs here: 1 2 3] and the output will be the sum is : 6.0
* Python has a large list of built-in libraries, anything else will need to be installed or it wont work
* Python can distribute looped workloads evenly across ranks

# Import the MPI module from mpi4py

from mpi4py import MPI

# Initialize the MPI communicator

comm = MPI.COMM\_WORLD

rank = comm.rank

size = comm.size

name = MPI.Get\_processor\_name()

total\_iterations = 12

for i in range(rank, total\_iterations, size):

print(f'hi from rank {rank}, iteration {i}')

# Your computation using i goes here

# Ensure all processes have completed before finalizing MPI

comm.Barrier()

MPI.Finalize()

* To be continued…