MPI & SLURM

Reid Ormseth

3 Aug 23

Running an MPI job

- How do you run a job on a basic cluster?

```
$ module load openmpi/4.1.4
$ cat machines
n1
n1
n2 slots=2
n3 slots=2
$ mpirun -np 6 --hostfile machines --prefix $MPIHOME ./a.out
Hello world from processor n1, rank 0 out of 6 processors
Hello world from processor n1, rank 1 out of 6 processors
Hello world from processor n3, rank 5 out of 6 processors
Hello world from processor n2, rank 2 out of 6 processors
Hello world from processor n2, rank 3 out of 6 processors
Hello world from processor n3, rank 4 out of 6 processors
```

- -np : Number of Processors
- --hostfile : Name of file with a list of nodes
  - May be specified multiple times, or with “slots=#” for multi-core
- --prefix : If MPI is not in your default env, tells mpirun where to find the MPI executables and libraries on the remote machines.

- ***Uses SSH to start processes on compute nodes***
  - You must have password-less SSH keys set up.
  - The other processes WILL NOT pick up current env variables (like $CWD).
  - It will look in your home directory and only find your exe & MPI if it is a system default, or you use –prefix.
  - You can use -x to pass env variables.
  - Use full paths, this example only works in your home dir!

Running an MPI Job with SLURM

- Simplified with ‘srun’:

```
$ salloc -N 3 -n 6
salloc: Granted job allocation 136
$ module load openmpi/4.1.4
$ srun --mpi=pmix ./a.out
Hello world from processor n3, rank 5 out of 6 processors
Hello world from processor n2, rank 3 out of 6 processors
Hello world from processor n2, rank 4 out of 6 processors
Hello world from processor n1, rank 0 out of 6 processors
Hello world from processor n1, rank 1 out of 6 processors
Hello world from processor n1, rank 2 out of 6 processors
```

```
$ module load comp/intel/2021.7.0 mpi/impi/2021.7.0
$ file `which mpirun`
/usr/local/intel/oneapi/2021/mpi/2021.7.0/bin/mpirun: POSIX
shell script, ASCII text executable
$ grep -A 1 SLURM `which mpirun`
    # SLURM
    if [ -n "$SLURM_JOBID" ]; then
        export I_MPI_HYDRA_BOOTSTRAP=slurm
```

- SLURM Assigns nodes to your job as soon as you start a job.
- Using ‘srun’ automatically detects the number of processes, nodes assigned, etc.
- **You can frequently still use ‘mpirun’, which will transparently call ‘srun’.**
  - Using mpirun helps portability to non-SLURM centers
- ***SRUN attempts to preserve your environment.***
  - srun leverages ‘slurmstepd’ on every compute node to launch jobs.
  - srun will capture your environment at time of submission and pass to all child processes.
  - Slurmstepd will configure things such as C-groups or attaching to the correct GPU.
  - If you load a module, those applications & libs will be available to all processes.
  - You can safely use relative paths in your application.

Three Ways to Run in SLURM

- salloc – runs a single command or gives you an interactive shell on a compute node:

```
[rormseth@discover21 h]$ salloc -N 1 --gres=gpu:4 --constraint=rome --partition=gpu_a100
salloc: Pending job allocation 20283361
salloc: job 20283361 queued and waiting for resources
salloc: job 20283361 has been allocated resources
salloc: Granted job allocation 20283361
salloc: Waiting for resource configuration
salloc: Nodes warpa008 are ready for job
[rormseth@warpa008 h]$
```

- sbatch – submits a script to run in the background:

```
[rormseth@discover21 h]$ sbatch hello.slurm
Submitted batch job 20283014
[rormseth@discover21 h]$ ls
20283014.o   20283014.e
```

- srun/mpirun – Launches parallel tasks, usually executed inside an ‘salloc’ or ‘sbatch’.
- Salloc is good for short, interactive access to compute nodes, particularly for compiling or post-processing. Otherwise, we highly recommend always using ’sbatch’.

Process Management

- By default, SLURM will allocate one task to every processor on a node
  - -N, --nodes
  - -n, --ntasks: how many total MPI processes to start
- Note that performance may be improved by not using all CPUs on a node.
  - --ntasks-per-node
- Can be used to allocate more memory by running fewer tasks than processors on node.
- OpenMP / MPI hybrid mode:
  - OpenMP is a programming technique to allow parallel processing via multiple threads in one process.
  - OpenMP is only single node, so need to use in conjunction with MPI
  - -N, -n, --cpus-per-task, plus set OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
  - Hybrid generally provides better performance than MPI alone at same scale.
- Many ways to specify the same result:

```
#SBATCH -G 48
#SBATCH --ntasks 48
#SBATCH -N 12
#SBATCH --constraint=rome
#SBATCH --partition=gpu_a100
```

```
#SBATCH --gres=gpu:48
#SBATCH -n 48
#SBATCH –ntasks-per-node 4
#SBATCH --constraint=rome
#SBATCH --partition=gpu_a100
```

Best Practices

- Always use sbatch, not ‘salloc’ or ‘srun’ to submit jobs, so you have a history of what you ran.
  - Don’t use cmd line options to sbatch, use ”#SBATCH –option” in the script
- Although sbatch will pull your current env (i.e., if you did a “module load intel”), always always put that in the batch so you can reproduce.
  - Start with a “module purge” in your sbatch script
  - Specify the version of compiler & MPI in module commands as the default will change.
- Use total number of tasks, not # of nodes.
- Use descriptive job names.
- Job output/error files with unique names- embed job ID into that filename.
- Drop an “env” into the top of your scripts, after you load modules, etc
- Throw a “-x” on the top of your shebang in sbatch scripts.
- Always specify a runtime limit

Best Practices

```
$ cat ihello.slurm
#!/bin/bash -x
#SBATCH -J intelTest
#SBATCH -n 6
#SBATCH -o %x.%j.o
#SBATCH -e %x.%j.e
#SBATCH --time=1:00:00

module purge
module load comp/intel/2021.7.0 mpi/impi/2021.7.0
env

EXE=‘./intelhello’
srun --mpi=pmix $EXE

$ sbatch ihello.slurm
Submitted batch job 23543249
$ tail -n 3 intelTest.23.e
+ return 0
+ echo ====================
+ srun --mpi=pmix ./intelhello
```

Packing & Replicated Jobs

- To run multiple jobs on a single node, use the Packable queue.
  - You will only share a node with other jobs you submit, not other users.

```
#SBATCH –p packable
```

- Job Arrays are for running large numbers of identical jobs.
  - This is more efficient than individual jobs, and frequently combined with “packable”.

```
$ cat runarray.slurm
#!/bin/bash -x
#SBATCH -J runarray
#SBATCH –p packable
#SBATCH -n 1
#SBATCH --array=0-6:2
#SBATCH -o %x.%A.%a.o
#SBATCH -e %x.%A.%a.e

env
module load impi intel
./myapp -input file.${SLURM_ARRAY_TASK_ID}
$ sbatch runarray.slurm
$ ls
file.0  file.4  runarray.25.0.e  runarray.25.2.e  runarray.25.4.e  runarray.25.6.e  runarray.slurm
file.2  file.6  runarray.25.0.o  runarray.25.2.o  runarray.25.4.o  runarray.25.6.o
```

Tips and Tools

- Log onto a node with a running job:

```
[rormseth@discover23 ~]$ srun --jobid=40652831 --pty bash
[rormseth@borgn181 ~]$ top
```

- To run on a specific type of node

```
#SBATCH --constraint=cas
```

Architecture	SLURM Constraint	CPUs/GPUs	Memory per CPU/GPU	Memory per Node*
Skylake	sky	40 CPUs (36 usable)	4 GB / CPU	192 GB
Cascade Lake	cas	48 CPUs (46 usable)	4 GB / CPU	192 GB
AMD Rome	rome	48 CPUs + 4 GPUs	100 GB / GPU	512 GB
AMD Milan	mil	128 CPUs	4 GB / CPU	512 GB

* This reflects physical memory, some amount is reserved by SLURM for OS, filesystem & overhead.

Recommended Tools

- Tests every SysAdmin & Power User should have close at hand:
- mpihello
  - C, C++ & Fortran examples available.
  - Used to test compiler & mpi config, user environment is correct, SLURM syntax and requests, job startup, etc.
- OSU MPI benchmarks [1]
  - osu_lat: Will test the latency between two nodes
  - osu_bw: Will test maximum bandwidth between two nodes
  - osu_mbw: Will test maximum bandwidth between a large number of nodes

1. https://mvapich.cse.ohio-state.edu/benchmarks/

Questions?

MPI Startup

- MPI Job startup and a basic ‘mpirun’
- srun vs mpirun
  - mpirun uses SSH, srun uses slurmstepd
  - mpirun needs details on node count, machinefile, etc., srun pulls from SLURM env
  - Most “mpirun” implementations are scripts that transparently call srun
- sbatch vs srun vs salloc
- Slurmd vs Slurmstepd, why slurmstepd?
  - Allow user onto a node (we don’t want to allow a user to SSH to someone else’s nodes)
  - Populate their environment to all nodes, and not a fresh login shell
  - Create custom C-groups for a step to run inside of
- How to open another shell on a node with my job to watch it?
- Good tests every admin/poweruser should have close at hand
  - mpihello
  - OSU MPI benchmarks, osu_lat, osu_bw, osu_mbw
- Processors vs Tasks vs Nodes
- Need more memory?
- MPI vs Hybrid OpenMP/MPI
- Job Packing
- SLURM job history and metrics
  - What jobs did I run?
  - Which succeeded and failed?
  - Where did they run?

Best Practices

- Always use sbatch, not ‘salloc’ or ‘srun’ directly, so you have a history of what you ran.
  - Don’t use cmd line options to sbatch, use ”#SBATCH –option” in the script
- Although sbatch will pull your current env (i.e., if you did a “module load intel”), always always put that in the batch so you can reproduce.
- Use total number of tasks, not # of nodes.
- Use descriptive job names.
- Job output/error files with unique names- embed job ID into that filename.
- Drop an “env” into the top of your scripts, after you load modules, etc
- Throw a “+x” on the top of your shebang in sbatch scripts.
- Always specify a runtime limit