How To Submit Parallel Jobs in O2

Wait! If you're breaking your job up into 100 pieces and running each one as a totally separate job (aka "embarrassingly parallel") you don't need the information on this page. Just run 100 *sbatch* commands. If you would like advice on writing a script to do those 100 *sbatch*, or information on job arrays - one *sbatch* command that submits 100 separate jobs - contact Research Computing.

O2 supports:

- shared memory parallelization on a single node, typically multithread, such as OpenMP
- distributed memory parallelization across multiple nodes, typically multitask such as MPI
- a combination of the two.

Very Important: *Submitting a parallel job will not make that job run parallel unless the application you are running explicitly supports parallel execution*. If you try to run a serial application as parallel you will always waste resources and might end up with a corrupted output.

Note for Orchestra Users:

Differently from Orchestra (our older cluster), in O2 we use a tool called *cgroup*s which constrains each job to have access **only** to the number of cores explicitly requested and therefore allocated by the scheduler on a given node. If a job seems to run slower on O2 than it did on Orchestra it is possible that in Orchestra the job was using more cores than what explicitly requested and this is not allowed on O2.

Shared Memory Parallelization

Many standard applications now have flags to run multithreaded. Threaded applications can use multiple cores on a node, but each multithreaded process/job requires access to the same physical memory on the node and it is therefore limited to run on a single node (i.e. shared memory). The most common type of multithread application uses the OpenMP application programming interface http://www.openmp.org

To submit a job that uses multiple cores, you must specify the number of job cores you want to use with the sbatch flag ***-c Ncores***. The maximum number of cores that can be requested with this approach is 20

The following example submits a job requesting 4 cores (all on the same node), a total of 10 GB of memory and a WallTime of 24 hours using the sbatch *--wrap=""* flag to pass the desired executable. The same job submission can be done using a script to pass slurm's flags and commands to be executed, see the main docs for more information.

```
sbatch -p medium -t 24:00:00 -c 4 --mem=10000
--wrap="your_multithreaded_application"
```

Note that each application may have a different way of specifying the number of threads to execute, for example bowtie uses -p to choose a number of threads. Please see the documentation for your particular application to see the appropriate flag, which is usually for a number of cores, nodes, cpus, or threads. You can also ask Research Computing for help with a particular application.

If your workflow does not contain any type of multithread or multitask parallelization but it starts multiple independent and cpu-consuming processes within the same job you should still request an appropriate number of cores using the above ***-c Ncores*** flag, (typically 1 core for every process). For example:

```
sbatch -p short -t 4:00:00 -c 2 --wrap="application_1 & application_2;
wait"
```

Note that in this case the command ***wait*** must be included at the end to ensure that all the applications started in parallel have finished.

This type of parallel job can be submitted to any of the available partitions (short,medium,long,interactive,priorirty) except the mpi partition.

Distributed Memory Parallelization

Many fewer programs allow distributed memory parallelization, although a number of important bioinformatics programs do use it to speed up execution for large analyses. This option is used for multitask applications capable of running in parallel over a distributed memory system where the allocated cores are not (necessarily) on the same node. The different tasks are usually capable of communicating via network with each other to exchange data, which is a necessary step since each task has only access to its own memory. In addition the parallel tasks in this type of application are usually initiated by a wrapper that interfaces with the scheduler to know all the different nodes where the cores were allocated.

The most common type of multitask application uses the MPI (Message Passing Interface) method. OpenMPI http://www.open-mpi.org/ is installed and available on the O2 cluster.

Preparation

OpenMPI uses `ssh` for intra-node communication, in order to allow `ssh` access between compute nodes without requiring a password, you must configure private and public keys. Host keys for all O2 compute nodes are already listed in `/etc/ssh/ssh_known_hosts`, so there is no need for you to add them to your `~/.ssh/known_hosts` file.

Please reference this wiki page for further instructions on creating SSH keys.

Submitting MPI jobs

To submit a MPI job on the O2 cluster you first need to load the gcc/6.2.0 and openmpi/2.0.1 or /3.1.0 modules

```
module load gcc/6.2.0

module load openmpi/2.0.1
or
module load openmpi/3.1.0
```

Similarly to the previous case to submit a multitask job you must specify the number of required tasks (i.e. cores) you want by using the flags ***sbatch -n*** **Ntasks** ***-p mpi ...*** . The maximum number of cores that can be requested with this approach is 640.

The following example is used to submit an mpi job requesting 64 cores and 3 days of WallTime.

```
$ sbatch -n 64 -p mpi -t 3-00:00:00 --wrap="ulimit -l unlimited; mpirun -np
64 your_mpi_application"
```

In a similar way it is possible to submit the same job passing the flags and the command with a script as in the example below:

```
$ sbatch myjob.sh
```

where myjob.sh contains:

```
#!/bin/bash
#SBATCH -n 64
#SBATCH -p mpi
#SBATCH -t 3-00:00:00
#SBATCH --mem-per-cpu=4000  ### Request ~4GB of memory for each core, total
of ~256GB (64 x ~4GB)

ulimit -l unlimited
mpirun -np 64 your_mpi_application
```

Note that for mpi jobs the **mpirun** command is required in order to start correctly the parallel pool of 64 tasks on the cores allocated across the different nodes.

It is also possible to request a specific number of cores per node using the flag ***--ntasks-per-node***. For example the following sbatch submission is used to request 40 cpu cores distributed over 10 nodes (4 cpus per node).

```
$ sbatch -n 40 --ntasks-per-node=4 --mem-per-cpu=4000 -p mpi -t 3-00:00:00
--wrap="ulimit -l unlimited; mpirun -np 40 your_mpi_application"
```

It is also possible to run mpi jobs on any of the other non-mpi partitions. In that case the maximum number of cpus that can be requested is 20.

***Important*:** The flag *--mem=XYZ* is used to request a given amount of XYZ memory *per each node used* independently by the number of cores that are dispatched on that nodes. Using *--mem=XYZ* when submitting jobs to the mpi partition can create an uneven memory allocation and consequently a long pending time and/or job failure due to insufficient memory. It is highly recommended to request the desired amount of memory as a "per core" using the flag *--mem-per-cpu=* when submitting distributed memory jobs as shown in the above examples.

It is also possible to combine the above flags with the flag **--mincpus=<n>** which can be used to specify the minimum number of CPU cores that needs to be allocated in each node, so for example the command:

```
$ sbatch -n 40 --mincpus=4 --mem-per-cpu=4000 -p mpi -t 3-00:00:00
--wrap="ulimit -l unlimited; mpirun -np 40 your_mpi_application"
```

would submit a job requesting 40 tasks (CPU cores) with a minimum of 4 CPUs per node.

Note: It is now recommended to add the command ***ulimit -l unlimited*** before the actual MPI command in order to fully enable IB connectivity.

Combining Distributed and Shared Memory Parallelization

A limited number of applications support a hybrid parallelization model where the MPI approach is used to distribute a number of tasks *Ntasks* acr oss different compute nodes and each of these task is then capable of running as a multithread process using *Nthreads* cores. In this scenario the total number of cores used by a job is the product of *Ntasks X* Nthreads and cores must be allocated accordingly. The following example shows how to submit a MPI + OpenMP combined job that initiate 10 mpi tasks and where each task runs as a multithreaded application using 4 cores for a total of 40 cores.

```
$ sbatch -n 10 -c 4 -p mpi -t 3-00:00:00 --wrap="ulimit -l unlimited;
mpirun -np 10 your_mpi+openmpi_application"
```