MEA 716- Tips for analysis on Henry2

1.26.2020, OIT-HPC

Topics:

- Where to log in, and where to run?
- Disk space management
- What NOT to do on the HPC system
- Creating an optimal batch submission script job for MPI WRF

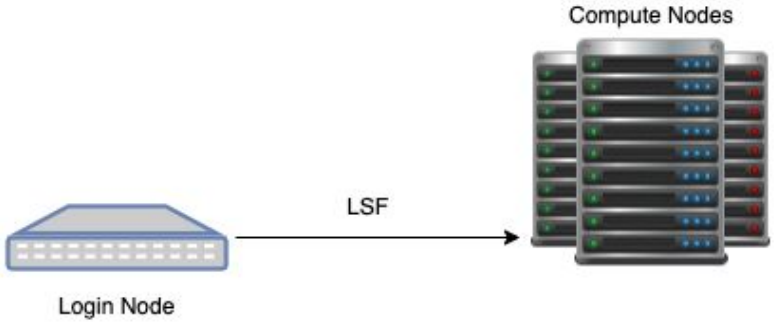
Where to log in, and where to run?

Options for logging in:

- Standard login node
- HPC-VCL node

Options for running:

- Batch job on compute nodes
- Interactive session on compute nodes
- HPC-VCL node



A **login node** is a node you log into. Once on a login node, you can access the HPC file system, and you can use LSF to submit jobs to compute nodes using 'bsub'.

Standard login node:

- There are 4 standard login nodes
- They each have 16 cores and 64GB RAM.
- At any given time, there could be a few, or a few dozen users on a single login node. You cannot do any 'work' on a login node.
- You use **ssh** to access the node.

HPC-VCL is also a login node:

- There are enough licences to accomodate ~30 concurrent users of the HPC-VCL.
- **VCL nodes have 4 (virtual) cores and 16GB RAM.**
- When reserving a session on HPC-VCL, you are reserving your own personal login node. You can work directly on that node.
- You use **RDP** to access the node

Compute nodes have up to 32 cores and up to 500GB RAM.

Login node vs HPC-VCL

Briefly:
ssh and **RDP** are both network protocols used to log into remote systems. It is possible to open remote displays with either protocol, but RDP is faster.

The purpose of an HPC-VCL login node is to **facilitate remote display** for HPC users, or allow users to **access the outside network** while doing calculations.

- The main reason to use HPC-VCL is that your application *requires* a GUI.
- Another reason is that you cannot use standard login nodes to compute, and you cannot access data from the campus network from a compute node, thus:
  - A **web browser** can only be used from the HPC-VCL
  - Any code/scripts that need to **access an outside database during execution** should be run on the HPC-VCL

Requiring interactivity is not the same as requiring a GUI

- For command line tasks or scripts not requiring a display, one can request an interactive session from a login node:

```
ssh login.hpc.ncsu.edu
bsub -Is -W 10 -n 1 tcsh
```

- You can use MATLAB, R, Python, NCO, NCL, CDO, etc. interactively from the command line
- MATLAB/Python/R scripts can be rewritten such that plots are written to a file rather than pop open a window.

Recommendations for this class

For analysis, you will use Jupyter notebooks, and do work that may require a GUI, and other interactive command line work. Go ahead and use the HPC-VCL for these. The class is small enough that we don't have to worry about license limitations. Still, only reserve the HPC-VCL for the required time, and when finished with work, delete that reservation. While you are on HPC-VCL, you can do "bsub" to submit WRF jobs or do interactive sessions on compute nodes as usual. Of course, If you are only submitting jobs or checking on them, it is certainly easier and faster just to ssh to a login node to do that than reserve HPC-VCL, open RDP, etc.

WARNING! Disk Quota Exceeded:

- Jupyter notebook opens a browser, mozilla. Mozilla saves a bunch of files to hidden directories (browser cache, etc.) which can fill your home directory.
- To display the disk usage in human readable form 1 directory deep:
  `du -h -d 1 .`
  The `.mozilla` and `.cache` directories are usually culprits, and they can usually be deleted.
- Before starting every program, cd into a scratch/work directory. Generally programs will write outputs and temporary files to the current working directory of where the program was called.

After this class

I recommending spending some time removing any part of your workflow that requires a display or downloads data. Gather data before the script is called, and submit scripts as batch jobs that create a directory of plots. You can use HPC-VCL to see the plots, or if possible, copy them via scp or Globus to a local machine for visualization purposes.

For a given session on HPC, should you use a standard login node, a HPC-VCL node, or interactive session on a compute node?

Session is only submitting and checking on WRF jobs

- ssh to standard login node

Session includes only data analysis that does not open windows/plots

- ssh to standard login node
- interactive session on a compute node

```
bsub -Is [options] tcsh
```

Session includes analysis that opens windows or uses a web browser,

- RDP into HPC-VCL node
- Do computations and open GUI applications directly in the terminal
- While you are on HPC-VCL, you can still do both `bsub` and `bsub -Is`

Data analysis that opens windows and has large RAM or CPU requirements

- RDP into HPC-VCL node
- `ssh -X` into a standard login node
- `bsub -Is` into a compute node

Data analysis with large RAM requirements requiring a web browser

- **Not possible** with standard setup
- (If you need that for your research, contact VCL about custom images)

**Data analysis with large RAM (or CPU) requirements requiring remote display (explanation)

Recall HPC-VCL nodes only have 16GB RAM, so you will need a compute node. You can use "bsub -Is" to request an interactive session while on HPC-VCL, but you cannot open a display there, and you will get an error:

```
[lllowe@n2c1-10 ~]$ display jobs.10pm.txt
No protocol specified
display: unable to open X server `vclhpc25:10.0' @
error/display.c/DisplayImageCommand/431.
```

So, with apologies, you need to:

- RDP into HPC-VCL node
- `ssh -X` into a standard login node
- `bsub -Is` into a compute node
- [do the stuff]

Then why don't I just 'ssh -X' to a standard login node from my local machine and skip the RDP part???

My understanding is that since VCL nodes are physically closer to standard login nodes than your local machine, it minimizes the ssh tunneling part of the process. I can't explain it better than that technically, but I have tested both and using use the RDP part first is much faster for real visualizations (something more requiring than opening a simple window).

One more thing

When requesting an interactive session with **`bsub -Is`**, you are on a compute node rather than a login node, but you still could be sharing that with others.

Use **`-x`** and/or other appropriate resource specifiers just as you would with a job submission script.

Documentation on specifying resources

Documentation on specifying memory

Disk space management

- This class has 5TB of space that is not purged, and you will be using that for pretty much everything. Delete output from test/garbage runs frequently if they are not already being overwritten by new runs, or after post processing is complete.

  `/gpfs_common/mea716_share`
- There is no 'normal' /share space for this class (the kind that is purged).
- Even though mea716_share is not purged, it is **not backed up**. Save scripts and other important small files to the /home directory.
- If you are developing your own Python/MATLAB/other scripts or code, consider creating a git repo to save and use version control. Any NCSU user can create repos on NCSU GitHub.

Disk management recommendations for your research group will be different, we can discuss if there is time at the end, or you can schedule a consultation.

Documentation on storage

What not to do on HPC

The most serious offenses are those that affect others sharing the system.

Staff priority is to identify issues that affect others, including

- Overuse of login nodes
- Overuse of compute nodes
- Misuse of the filesystem

Also

Currently we have limited resources to *identify and reach out* to each user who appears to be underutilizing resources because of inefficiency. But we do have time to consult with users who *reach out to us*!

HPC Staff are happy to give consultations to

- Help you to both avoid doing bad things while doing work on HPC
- Help you find the best and most optimal ways to do your work on HPC

Overuse of a Login node (multiple simultaneous file transfers, excess CPU load, or using all memory)

Best case:

- Michael sees that and sends you an email to request you stop [doing that]

Worst case:

- You can bog down the node, making it slow for all current users.
- You could cause the node to crash, resulting in users losing contact with the node, losing work (edits to files, compilation steps, etc.)

How it happens? (besides skipping HPC training altogether)

- Users 'time out' of an interactive session without noticing
- Users think a small test directly on a login node won't hurt anything
- Compilations that use parallel make based on number of available processors (make -j)

Overuse of a compute node (too much CPU or memory)
Best case:

- Your job, and jobs of others sharing the node, runs slower than optimal.

Worst case:

- The node becomes unresponsive or crashes, killing your job and jobs of those sharing the node.

Use of -x is good, but:

- An unresponsive node has to be rebooted, so even though only your job was killed, it still results in loss of availability of compute resources for all users.

How it happens?

- User has no idea about the memory requirements of the application, or doesn't realize different nodes have varying amounts of RAM (some nodes have less RAM or slower clock speeds than your laptop)
- User doesn't know their job automatically spawns threads equal to available cores, or uses multithreaded libraries
- User with shared memory application does not request xhost=1
- Application not properly installed with supported MPI, tasks not distributed properly
- User properly requests a number of MPI tasks equal to the number of cores on the node (**X**), but not realizing each of those tasks will spawn threads equal to the number of cores on the node, resulting in a load of **X***__X__.

Misuse of disk space

- Login and compute nodes are only shared transiently by a few to a dozen users.
- The FILE SYSTEM affects every user for long periods of time.
- Problems in usage may cause jobs to fail clusterwide - input data becomes unavailable, and there is no space for jobs to write output data.

Two issues

1) **Disk quota exceeded (beyond your /home)**
   - **Actual disk usage:** To avoid affecting your teammates, keep track of your disk usage with 'du', or check the 10 TB quota for /share for your research group with

     ```
     module load apps
     hpc-quota
     ```

   - **Inodes:** Your group's '/share/$GROUP' is a subset of a partition that shared by other groups. There is also a quota for the number of files (inodes) on those partitions. Avoid affecting others by **not** creating tons of files (ton=10k+ files per run, and the most problematic users can create 100k or millions of files)

2) **Stressing the file system by doing things not agreeable to a parallel file system...**

2) Stressing the file system by doing things not conducive to a parallel file system

We have a shared, parallel file system, optimal for writing large data files and accessing chunks data simultaneously from different compute nodes. Data is distributed over multiple storage servers, it is replicated over multiple servers, and it needs to keep track of changes in that data to keep it consistent across servers.

Thus, do not:

- Create a small file or files, then do many small file operations [e.g. echo > file, cat >> file, mv file, rm file] many times in succession
- And especially don't do that multiple nested loops
- And *especially* don't submit multiple jobs doing that at the same time

(See news item for Feb. 1, 2019)

Q) What if a user's code needs to [do odd file manipulations or generate millions of files], does it mean they can't use HPC?

- No. In most cases, I can recommend a way for a user to do what they need without [doing the odd thing]. If I cannot find a way to do that on HPC, we will point you to a more appropriate resource.

Optimizing batch scripts for WRF

Goal: Minimize time to results.
Manipulate resource requests (cores, RAM, ptile, etc.) in order to:

- Minimize Code execution time
  and
- Minimize Queue wait time

When doing this, consider the queue wait and resource availability for others as well.

Before spending effort on optimization, you need to do basic profiling:

1) Find an example case, preferably something that fits comfortably on one node (RAM-wise).
2) Run the case for a small number of timesteps, but not so small that initialization would dominate or that it excludes writing of output files. Start with 8 MPI tasks (arbitrary suggestion). Run using "-x" to avoid interference from other jobs. (**mea516 does not allow -x, discuss)
3) Look at the LSF output file: to check run time, basic memory requirements, and CPU utilization.

Recommended: add `/usr/bin/time` before `mpirun` in your batch script for additional info beyond what is given by LSF.

Using LSF output to inform memory and time

```
Resource usage summary:

    CPU time :                       209542.33 sec.    (1)
    Max Memory :                     15725.28 MB       (2)
    Average Memory :                 10982.08 MB
    Total Requested Memory :         -
    Delta Memory :                   -
    Max Swap :                       17773 MB
    Max Processes :                  4
    Max Threads :                    38
    Run time :                       52134 sec.        (3)
    Turnaround time :                52125 sec.
```

(1) CPU time usually should be the wall clock time elapsed times number of cores

(2) Max memory is how you will determine how much memory to request in the job script

(3) Run time is wall clock time - use this to determine how much time to request in the job script

Run test case with different resources

Starting with your test case that was run on one node

- First check memory and swap. Request a node with more memory and redo the test if RAM requirements were close to the max RAM available on the node.
- Change the number of MPI tasks used, and note the run time for each.
- Important - use resources specifiers such that you get the same model nodes for each test! Link: LSF Resources, Memory

It is hard for me to write up specifics, so instead, let's see a case study.

The following is a quick and dirty test to get some preliminary scaling results before the holiday break.

Excerpts from my notes from profiling a ROMS model on Stampede2

12.22.2020
Timing tests:

48,6x8= 15:30.45	15.5/5.51 = 1	(1)
48,8x6= 16:04.63		
96= 7:03.98	15.5/7 = 2.2	(2)
192= 4:11.75	15.5/4.2 = 3.7	(4)
384= 2:24.25	15.5/2.4 = 6.5	(8)
768 = 1:40.28	15.5/1.67 = 9.3	(16)
1536 = 1:53.29	15.5/1.88 = 8.6	(32)

← **Original tests**
The ROMS log indicated a disproportionate amount of time was spent in I/O for the 1536 core run, so perhaps code would scale if parallel netcdf was enabled at compile time.

1.21.2021
Redo serial:
module load netcdf
/usr/bin/time ibrun ./coawstM.serialIO ocean_mobile_whole.in > log_whole.out
mkdir serialIO_96
7225.98user 126.41system 5:09.26elapsed 2377%CPU (0avgtext+0avgdata 343204maxresident)k

In the test to see if parallel I/O improved performance, I saw time go from **7** minutes to **5** minutes. But then I **reran** the serial I/O case and also got 5 minutes...hmmm

I redid with the old executable and also got 5min.

1.21.2021
Seems I was using twice the nodes than in the first tests, so they aren't comparable. Try again:
#SBATCH -N 2 # Total # of nodes
#SBATCH -n 96 # Total # of mpi tasks
And now it is the same time: 7:13

My submit files were different...I was using different amount of resources!

- **It is very important to be systematic, document things, keep notes, and save not only log files but also submit scripts for each job.**
- **Also, for 'official' scaling tests, you should perform the tests several times for each case.**

If the code used 96 MPI tasks in each case, why was running on 4 nodes faster than running on 2?

I was running on nodes with 48 cores. Runs used 96 MPI tasks, either using 2 nodes with 48 tasks on each node, or using 4 nodes with 24 tasks on each node.

Most likely reason for speedup:

- Using 4 nodes, each task had additional memory available (***cache and all that)**

Other possible reasons:

- In principal, 96 MPI tasks use 96 cores, and the actual CPU utilization should be 96, but if a code has OpenMP enabled, or is using threaded libraries, then it *could* be getting some use out of those extra cores. Your WRF does not have OpenMP enabled, it was compiled MPI only.

Tangent:

- If MPI communication dominates, it is possible to get worse performance by using more nodes for a given task count, because of inter-node communication...but I find that unless the code is really bad, it usually doesn't affect the time very much on this scale. There are many profilers to check time spent in MPI calls.

***cache and all that

My non-technical way of explaining how execution time is affected by cache…

How long does it take you to call someone on the phone if…

1) You have the number memorized (L1 cache)
2) The number is in your notepad on your desk (L2 cache)
3) The number is in a phonebook on your desk (L3 cache)
4) The number is in a phonebook on your desk, but your friend comes over and borrows it from you, then you have to go to their desk to take it back… (cache miss/contention)
5) The phonebook is in the library (code starts to read/write from disk...only speed wise analogy...not good analogy for what is actually happening!)

See for technical explanations:

https://www.makeuseof.com/tag/what-is-cpu-cache/

Basic Performance testing

Make efficient use of resources

Q: Does using more resources make your code run faster?

1) No

   Then don't use more resources.

2) Yes
   a) Do you *need* it to be faster? If so, how fast do you need it?
      i) If you aren't going to have time to look at the output until tomorrow, it's probably okay if it takes 8 hours instead of 2.
      ii) If you have to wait in the queue longer, is it worth it to you to ask for more resources?
      iii) Do you need it to be faster to fit within Wall Clock limit of queue?
   b) Is the speed increase worth the resource increase?
      i) Look at answer to 2a
      ii) Look at Ahmdahl's law

Ahmdahl's Law

If you have a code that is *perfectly parallel*...then using 100 processors gives you 100 times speed up.

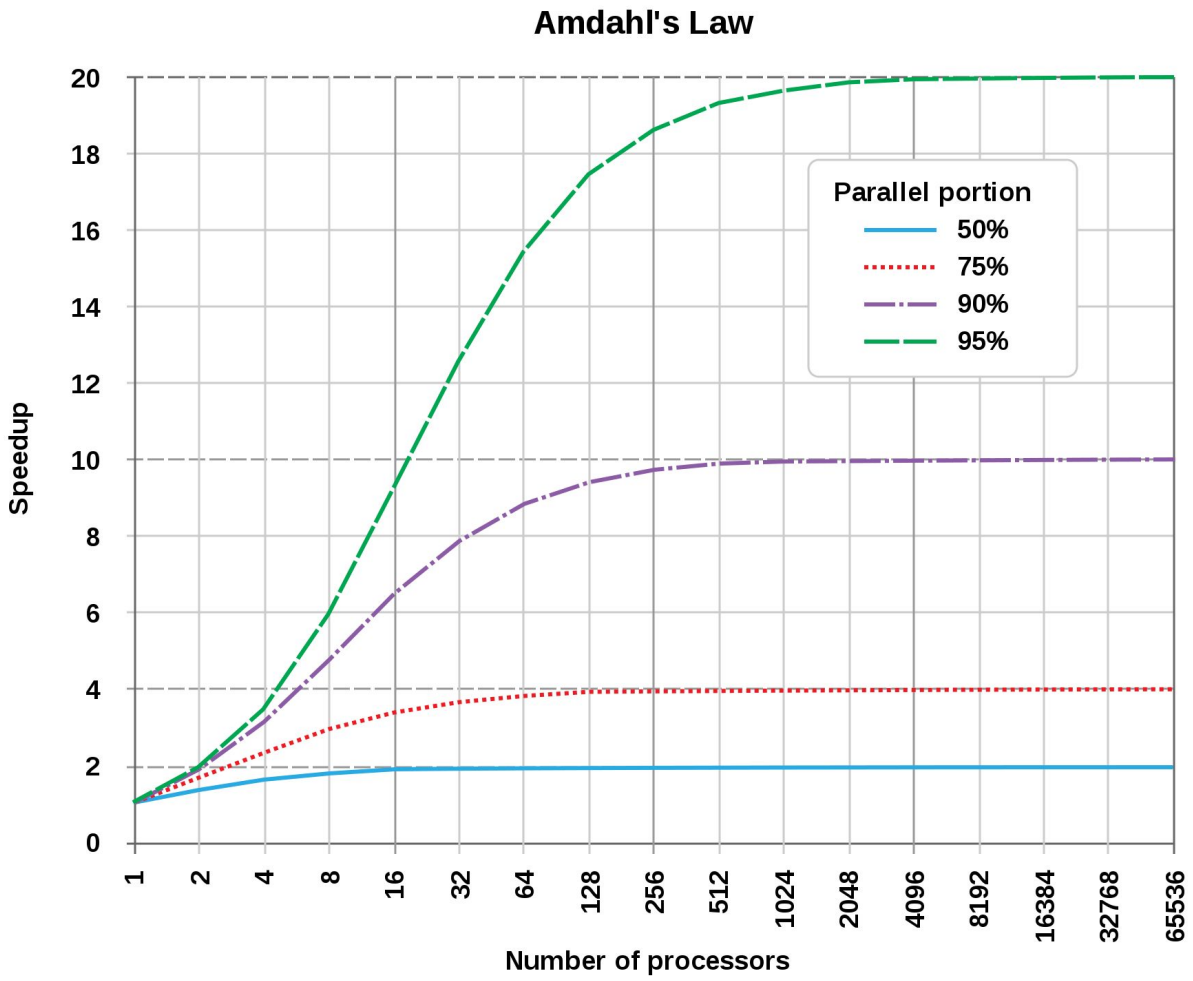
It's probably not perfectly parallel. In that case, the non-parallelizable part of your code determines your speedup. If your code is half parallel, your best speedup will be 2.

The next slide shows the maximum theoretical speedup for codes with varying degrees of percent parallel.

Notice that after a certain point, even though the code does continue to get faster, the speedup is not worth the additional resources.

Also note that if your code scales well, it probably scales up. (Increasing the number of processors is worth it as the problem size is increased.)

Here is a picture of **PERFECT** scaling:



By Daniels220 at English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6

Relating this to WRF

Your scaling test will give you an idea of time and memory requirements for grid of a certain size.

For a given grid:

- There will be an optimal number of (maximum) cores to run on
- Running twice as many timesteps will take roughly about twice as long, given the same or similar model processors

Sizing up - If the code scales well, and I assume WRF does:

- If you double the problem size, you can probably double the number of cores requested and roughly have the same runtime/efficiency
- Memory is based on number of gridpoints (and variables, if it might be different for different runs. So doubling total number of gridpoints will double the memory requirement...but if you double cores, the memory required per core is roughly the same
- Notice how many times I said 'roughly'...Do a test with a few timesteps, don't just change parameters and submit a many-hour job.

How does this relate to creating the batch script???

LSF parameters to change include

- Queue `-q queuename`
- Number of cores `-n X`
- Span (number of cores per node) `-R span[ptile=X]`
- Memory required per node `-R "rusage[mem=X]"`

Specifying Queue - don't, unless...

ccee cores and memory per node

8 cores 24 GB	34 nodes
12 cores 48 GB	35 nodes
16 cores 128 GB	6 nodes
20 cores 128 GB	42 nodes
24 cores 128 GB	32 nodes
32 cores 192 GB	88 nodes

Priority: 100

Most recent hardware are mainly confined to Partner queues (for recent Partners)

Use, if not filled by other group members

mea716 cores and memory per node

8 cores 16 GB	114 nodes
8 cores 48 GB	3 nodes
8 cores 24 GB	84 nodes
12 cores 24 GB	7 nodes
12 cores 48 GB	58 nodes
12 cores 36 GB	27 nodes
16 cores 72 GB	51 nodes
16 cores 56 GB	1 nodes
16 cores 64 GB	63 nodes
20 cores 128 GB	33 nodes

Priority: 90

Use if you need elevated priority, if not filled by classmates

standard_ib cores and memory per node

8 cores 24 GB	34 nodes
12 cores 48 GB	61 nodes
16 cores 128 GB	6 nodes
20 cores 128 GB	42 nodes

Priority: 64

Use it if you need infiniband

Usually, just let LSF pick a default queue

single_chassis cores and memory per node

8 cores 16 GB	156 nodes
8 cores 48 GB	3 nodes
8 cores 24 GB	176 nodes
12 cores 36 GB	31 nodes
12 cores 24 GB	7 nodes
12 cores 48 GB	105 nodes
16 cores 72 GB	35 nodes
16 cores 56 GB	1 nodes
16 cores 128 GB	5 nodes
16 cores 64 GB	42 nodes
20 cores 512 GB	2 nodes
20 cores 256 GB	2 nodes
20 cores 128 GB	50 nodes
24 cores 128 GB	21 nodes
32 cores 192 GB	28 nodes

Priority: 55

Default...no need to specify unless you are in cos queue by default

serial cores and memory per node

8 cores 16 GB	42 nodes
8 cores 24 GB	107 nodes
12 cores 48 GB	65 nodes
16 cores 128 GB	4 nodes
16 cores 72 GB	6 nodes
16 cores 64 GB	8 nodes
20 cores 128 GB	7 nodes
20 cores 512 GB	1 nodes
20 cores 256 GB	1 nodes

Priority: 30

Priority is given to parallelized jobs.
BTW, serial_long is much worse…

Default...no need to specify unless you are in cos queue by default

Note: this information can change without notice...avoid creating an LSF script that depends on which nodes are in a queue.

Specifying number of tasks, when tasks=cores

```
#BSUB -n [#tasks]
```

Choose **n** based on scaling tests

Just specify **n** with no other modifiers if

- Each task requires a small amount of memory
- Communication doesn't matter
- Sharing a node with others doesn't matter (i.e., you don't care if another job slows down the node, or even crashes it)

Cases where you don't care are things like MCMC, where tasks are largely independent and require minimal CPU and RAM.

You usually care for gridded codes...i.e. WRF

Specifying span and/or memory

```
#BSUB -n [#tasks]
#BSUB -R span[ptile=#tasks_per_node]
and
#BSUB -x
or
#BSUB -R "rusage[mem=X]"
```

For gridded codes like WRF

- You want the tasks assigned to cores in a logical way to optimize MPI communication
- You don't want another job competing for CPU and cache (memory)
- You might want extra RAM per task, rather than using every single core on the node

It is optimal to specify how many tasks you want assigned to each node, and request exclusive use of the node. If you don't specify **`-x`**, you could be sharing any unreserved cores with another job (possible one of your own jobs.

- If you just want to ensure there is enough memory on each node, instead of using **`-x`**, you can specify ptile ***and*** minimum memory required on each node...and if you don't care if another user might use more than their fair share of the memory on your node.