Quickstart Guide

The much longer, detailed quickstart guide follows this lightning-quick quide.

Lightning-quick Quickstart Guide

1. Use lalloc <Number of nodes> to get an allocation.

lalloc 2 -W 30 -q pdebug : 2 nodes for 30 minutes in queue pdebug

2. Check your node(s) using **check_sierra_nodes**

3. **lrun -T1 hostname | sort** gets you the list of nodes you got

4. Launch compiles using **make -j**

5. Run job using lrun:

lrun -T<ntasks_per_node>|-p<ntasks>|-n<ntasks> [-N<nnodes>] [--nolbind] <some jsrun options> <app> [app-args]

set environment variable **MPIBIND to `j`** to see jsrun invocation that lrun generates.

6. The easy way is to specify tasks per node and let lrun figure out the number of ranks from the allocation: **lrun -T4 hello.exe** will run 4 ranks in a 1 node allocation and 16 on a 4 node allocation

7. Use the **-M"-gpu"** option to use GPUDirect with device or managed memory buffers

8. **lsfjobs** to find the state of the job queue

9. The **-m "launch_hosts sierra24"** option of bsub requests a particular node or nodes

10. To submit a 2048 node job to the pbatch queue with core isolation and 4 ranks per node for 24 hours:

bsub -nnodes 2048 -W 24:00 -G guests -core_isolation 2 -q pbatch lrun -T4 <executable> <args>

Detailed Quickstart Guide

NOTE: **SIERRA**, **LASSEN**, and **RZANSEL** are in pretty good shape as of October 10th, 2018 but some key software features we want are scheduled to be delivered between now and Spring 2019, so things are still changing rapidly. We are updating software with new versions daily until probably February 2019, so these are not yet production clusters with the stability that implies. Please tell us about problems so we can fix them!

Here is some more detailed instructions that are likely to go out of date quickly (updated October 10th, 2018). John Gyllenhaal (gyllen@llnl.gov, (925) 424-5485) is the point of contact for Sierra during its bring up phase.

CONTENTS

Login from somewhere inside LLNL

You cannot ssh directly to SIERRA or LASSEN from offsite yet. Login from somewhere inside LLNL (like rzgenie, oslic, or your desktop). If you are offsite, I recommend you ssh to oslic first.

oslic8{gyllen}2: **ssh sierra**

REQUIREMENT: You must have passphrase-less ssh keys set up on sierra

There are instructions on how to set up passphrase-less ssh keys at the bottom of this page. You must have passphrase-less ssh keys set up on sierra, lassen or rzansel (in your LC CZ ~/.ssh directory) in order to successfully run jsrun or lrun on sierra. This requirement is expected until at least January 2019. The bsub/lalloc/jsrun/lrun commands currently uses ssh under the covers to launch daemons and you will get a variety of cryptic error messages if you do not have passphrase-less ssh keys set up (**ssh-agent is NOT supported** in this mode). Many error messages are possible if you do not have passphrase-less ssh keys set up but one common one is: **Error initializing RM connection. Exiting.**

First time SIERRA/LASSEN/RZANSEL users should verify they have passphrase-less ssh keys set up.

Try sshing into lassen from lassen, sierra from sierra, or rzansel from rzansel after running kdestroy (destroying any kerberos tickets first so you don't get false indications that your ssh keys are set up properly):

sierra4367{gyllen}13: **kdestroy**

sierra4367{gyllen14: **ssh sierra**
Warning: Permanently added 'sierra,134.9.39.31' (ECDSA) to the list of known hosts.
Password:

If you see a Password: or Passphrase: prompt, jsrun and lrun will not work (since you don't have passphrase-less ssh keys) and will give many cryptic error messages (including **Error initializing RM connection.**)! If you believe you have ssh keys set up and they are not working, make sure ~ and ~/.ssh are not writable by anyone but yourself and ~/.ssh/authorized_keys exists and has the appropriate public key(s) in it. **The re are instructions on how to set up passphrase-less ssh keys at the bottom of this page.**

ssh key agents breaks these machines but passes the above tests. You must use passphrase-less ssh keys right now on CORAL systems.

If you can ssh into machines but bsub/lalloc/jsrun/lrun still doesn't work, **make sure your .ssh/config file does not default to port 622**. As of 8/2/2018, RZANSEL launches hosts with ssh with a host name that doesn't match rzansel* (perhaps an ip address).

Use ‘lsfjobs’ to see machine state

Use ‘lsfjobs’ to see what is running, what is queued and what is available on the machine:

sierra4368{gyllen}13: **lsfjobs**

```
<snip>
*********************************************************************************************
* QUEUE           NODE GROUP      Total   Down    Busy   Free  NODES                        *
*********************************************************************************************
   -              debug_hosts        36      1       0     35  sierra[361-396]
   -              batch_hosts       871     14     212    645
sierra[397-531,533-612,631-684,703-720,1081-1170,1189-1440,1819-2060]
<snip>
```

Allocate interactive nodes with 'lalloc'

Use the LLNL-specific 'lalloc' bsub wrapper script to facilitate interactive allocations on CORAL and CORAL EA systems. The first and only required argument is the number of nodes you want followed by optional bsub arguments to pick queue, length of the allocation, etcetera. Run 'lalloc' with no arguments for usage info.

As of Oct 23th, SIERRA, LASSEN, and RZANSEL are again currently running lalloc/2.0, which uses 'lexec' to place the shell for the interactive allocation on the first compute node of the allocation. The CORAL EA systems (RZMANTA, RAY, SHARK) currently always put you on the first compute node (which lalloc/2.0 emulates for CORAL systems).

The lalloc script prints out the exact bsub line used. For example, 'lalloc 2' will give you 2 nodes with those listed defaults:

```
lassen708{gyllen}2: lalloc 2
+ exec bsub -nnodes 2 -Is -XF -W 60 -G guests -core_isolation 2 /usr/tce/packages/lalloc/lalloc-2.0/bin/lexec
Job <3564> is submitted to default queue <pbatch>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on lassen710>>
<<Redirecting to compute node lassen90, setting up as private launch node>>
```

Here is the current usage info for lalloc/2.0 as of Oct 10th, 2018:

```
Usage: lalloc #nodes <--shared-launch> <supported bsub options> <command>
  Allocates nodes interactively on LLNL's CORAL and CORAL EA systems
  and executes a shell, or the optional <command>, on the first compute node
  (which is set up as a private launch node) instead of a shared launch node

lalloc specific options:
  --shared-launch     Use shared launch node instead of a private launch node

Supported bsub options:
   -W minutes          Allocation time in minutes (default: 60)
   -q queue            Queue to use (default: system default queue)
   -core_isolation #   Cores per socket used for system processes (default: 2)
   -G group            Bsub fairshare scheduling group (default: guests)
   -Is|-Ip|-I<x>       Interactive job mode (default: -Is)
   -XF                  X11 forwarding (default if DISPLAY set)

  Example usage:
     lalloc 2        (Gives interactive shell with 2 nodes and above defaults)
     lalloc 1 make -j (Run parallel make on private launch node)
     lalloc 4 -W 360 -q pbatch lrun -n 8 ./parallel_app -o run.out

  Please report issues or missing bsub options you need supported to
  John Gyllenhaal (gyllen@llnl.gov, 4-5485)
```

Known lalloc/2.0 issues

As of Oct 10th, 2018, there are two known lalloc/2.0 issues with running on the first backend node.

1) Killing an lrun/jsrun with control-C causes the allocation daemons to die, preventing future lrun/jsrun invocations from working (gives messages like: Could not find the contact information for the JSM daemon. and: Error initializing RM connection. Exiting.). You must exit the lalloc shell and do another lalloc to get a working allocation. IBM has reproduced and we expect this to be fixed in Nov 2018.

2) Many GPU programs (gvim, memcheckview, etc.) have a 12 second delay the first time they are invoked. Future invocations in the same allocation work fine. Sometimes, the allocation doesn't exit properly after typing 'exit' until control-C is hit. This is caused the the startup of dbus-daemon, which is commonly used by graphics programs. We are still exploring solutions to this.

Allocating interactive nodes directly with 'bsub' (with core isolation 2) and lexec

You can run the same bsub command shown by 'lalloc' to get an interactive allocation. The last portion of the bsub lines is the executable to run and lalloc/2.0 uses 'lexec' to place your shell (or optional command like 'make') on the backend node. Here is how you ask for 16 nodes from pbatch with 60 minutes run time and X11 support (**only do -XF if DISPLAY is valid**). You have to use **-nnodes <nodes>** instead of -n <cores> with bsub on sierra, rzansel, and lassen. All other bsub options are the same as the CORAL EA systems.

rzansel61{gyllen}3: **bsub -nnodes 2 -Is -XF -W 60 -G guests -core_isolation 2 lexec**
Job <14698> is submitted to default queue <pdebug>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on rzansel62>>
<<Redirecting to compute node rzansel16, setting up as private launch node>>
rzansel16{gyllen}2:

You end up in **pbatch if -q not specified**. To select the pdebug queue, use **-q pdebug** on the bsub line instead. lsfjobs lists the available queues at the bottom of its listing. Please do not use anything other than pbatch or pdebug unless you have been directed to do so.

Enabling core isolation with bsub -core_isolation 2

As of Aug 2, 2018, it is now possible to enable core isolation on SIERRA and BUTTE by adding '-core_isolation 2' to the bsub line or disable core isolation by specifying '-core_isolation 0' . For example:

bsub -nnodes 4 **-core_isolation 2** -Is -XF -W 60 -G guests /bin/tcsh

This isolates all the system processes (including GPFS daemons) to 4 cores per node (2 per socket). With 4 cores per node dedicated for system processes, we believe there should be relatively little impact on GPFS performance (except perhaps if you are running ior).

Please be aware that using -core_isolation 2 causes jsrun to isolate your MPI tasks with cgroups to *exactly* and *only* what you ask for. For example, if '-g #' is not used to specify the number of GPUs per task on the jsrun line, NO GPUs will be visible to your MPI task! Similarly, if '-c #' is not used to specify the number of CPUs per task on the jsrun line, only 1 cpu will be visible to each MPI task!

Submitting non-interactive bsub scripts from scripts

It is often useful to have a script that submits bsub scripts for you. However, bsub currently only accepts #BSUB options when piped in on stdin. It is often convenient to use the 'cat << EOF' trick to embed the bsub script you wish to pipe in to stdin in your script. This was extracted from the July 24th, 2018 getting started tutorial https://computing.llnl.gov/tutorials/SierraGettingStarted/5.BuildingRunningJobs.pdf to provide a place to get started:

```
sierra4359{gyllen}52: cat do_simple_bsub
#!/bin/sh
cat << EOF | bsub -nnodes 32 -W 360
#!/bin/bash <- optionally set shell language, bash default
#BSUB -core_isolation 2 -G guests -J "MYJOB1"
cd ~/debug/hasgpu
lrun -T 4 ./mpihasgpu arg1 arg2
EOF

sierra4359{gyllen}53: ./do_simple_bsub
Job <143505> is submitted to default queue .
```

Don't slam the SHARED BATCH LAUNCH NODE that bsub runs your script on

Unlike with SLURM, bsub runs your script/shell on one of the five SHARED BATCH LAUNCH NODEs, essentially another login node. You need to run jsrun/lrun under an allocation on the launch node. You will get an error about jsrun not found otherwise. Since the launch nodes is a shared resource, try not to abuse it too much (like BG/Q, it is not dedicated to you (unlike our x86 clusters)). With '**lexec**' you can easily run commands on the first backend node, like 'lexec make -j' or 'lexec spack install'. We are investigating wrapping bsub so that the script runs on the first compute node but that is not available yet.

Make sure to use 'bkill jobid' on any jobs that appears hung or not finishing.

As of April 24th, 2018 we believe we have resolved the NFS issues breaking job launches to random nodes. However, if anything goes wrong, we still recommend exiting your allocation and relaunching a few minutes later. At this point, any 'bad' node on sierra can cause problem for the entire system. We now aggressively look for bad nodes at the end of jobs, so by using 'bkill' on all your hung/bad jobs, we should automatically drain those bad nodes to prevent them from causing problems. Note that 'bkill 0' will kill ALL of your running and queued jobs.

Use 'check_sierra_nodes' to verify you have good nodes

You can also hunt for bad nodes fairly quickly by running check_sierra_nodes with no arguments before your runs, which give output in this form:

sierra4367{gyllen}4: **check_sierra_nodes**
STARTED: 'jsrun -r 1 -g 4 test_sierra_node -mpi -q' at Fri Apr 27 16:38:58 PDT 2018
SUCCESS: Returned 0 (all, including MPI, tests passed) at Fri Apr 27 16:39:03 PDT 2018

The last line will start with SUCCESS if no bad nodes were found and the return code will be 0.

NOTE: Some of the bad node states cause MPI_Init to hang. The 'check_sierra_nodes' runs in approximately 6 seconds on a low number of nodes and in about 70 seconds on 650 nodes. If your check_sierra_nodes invocation takes more than two minutes (on 650 or less nodes), that also indicates an error condition that needs to be reported.

Please send any FAILED messages to John Gyllenhaal (gyllen@llnl.gov), so we can drain or fix the bad nodes. The messages will be of the form:
FAILED: sierra648 GPU 2 failed cudaMalloc 1: all CUDA-capable devices are busy or unavailable

The key for running successfully on Sierra right now is having no bad nodes in your allocation.

P. S. Running 'check_sierra_nodes -h' provides options for more verbosity and allows for not checking MPI. The auxiliary 'test_sierra_node' program (which is used in the epilogue and we have placed a copy in /usr/tcetmp/bin) can also be run directly on backend sierra nodes to test for problems.

Running MPI jobs or compilations on backend compute nodes with 'lrun'

If you are running just 1 MPI job in an allocation, we recommend you use the LC-written 'lrun' wrapper for jsrun, instead of using jsrun directly to launch jobs on the backend compute nodes. Currently lrun does not support running multiple jobs in an allocation, so you need to use jsrun directory. We are investigating writing another script (perhaps called lpack) for running multiple MPI jobs in an allocation.

Usage: **lrun -T<ntasks_per_node>|-p<ntasks>|-n<ntasks> [-N<nnodes>] [--nolbind] <some jsrun options> <app> [app-args]**

The **lrun** wrapper command line requires either **-p<tasks>** or **-n<tasks>** or **-T<tasks_per_node>** to be specified. By default, all nodes in the allocation will be used but a subset may be used by optionally specifying **-N<nodes>**. The 'lrun' wrapper then runs your parallel job with pretty good cpu, memory, and GPU binding across the nodes, unless **--nolbind** is specified. As of 4/27, lrun supports running with core isolation (i.e., lalloc or bsub -cores_isolation 2) and should yield similar binding/unbound behavior.

Note: **lrun is designed for running exactly one job on an allocation at a time**. If you wish to run multiple simultaneous jobs in an allocation, you must use jsrun directly.

Examples of using lrun to run MPI jobs

Note: As of July 2018, jsm now includes the utility program '**js_task_info**' that provides great binding and mapping info, but it is quite verbose. I am removing much of the output and replacing it with '...' for readability.

If you have a 16 node allocation, you can restrict the nodes lrun uses with the -N <nodes> option, for example, on one node:

sierra4367{gyllen}9: **lrun -N 1 -n 4 js_task_info | & sort**
Task 0 ... cpu[s] 0,4,... on host sierra1301 with OMP_NUM_THREADS=10 and with OMP_PLACES={0},{4},... and CUDA_VISIBLE_DEVICES=0
Task 1 ... cpu[s] 40,44,... on host sierra1301 with OMP_NUM_THREADS=10 and with OMP_PLACES={40},{44},... and CUDA_VISIBLE_DEVICES=1
Task 2 ... cpu[s] 88,92,... on host sierra1301 with OMP_NUM_THREADS=10 and with OMP_PLACES={88},{92},... and CUDA_VISIBLE_DEVICES=2
Task 3 ... cpu[s] 128,132,... on host sierra1301 with OMP_NUM_THREADS=10 and with OMP_PLACES={128},{132},... and CUDA_VISIBLE_DEVICES=3

All these examples do binding, since --nolbind was not specified.

sierra4367{gyllen}11: **lrun -N 3 -n 6 js_task_info | & sort**

Task 0 ... cpu[s] 0,4,... on host sierra1301 with OMP_NUM_THREADS=20 and with OMP_PLACES={0},{4},... and CUDA_VISIBLE_DEVICES=0 1
Task 1 ... cpu[s] 88,92,... on host sierra1301 with OMP_NUM_THREADS=20 and with OMP_PLACES={88},{92},... and CUDA_VISIBLE_DEVICES=2 3
Task 2 ... cpu[s] 0,4,... on host sierra1302 with OMP_NUM_THREADS=20 and with OMP_PLACES={0},{4},... and CUDA_VISIBLE_DEVICES=0 1
Task 3 ... cpu[s] 88,92,... on host sierra1302 with OMP_NUM_THREADS=20 and with OMP_PLACES={88},{92},... and CUDA_VISIBLE_DEVICES=2 3

Task 4 ... cpu[s] 0,4,... on host sierra1303 with OMP_NUM_THREADS=20 and with OMP_PLACES={0},{4},... and CUDA_VISIBLE_DEVICES=0 1
lTask 5 ... cpu[s] 88,92,... on host sierra1303 with OMP_NUM_THREADS=20 and with OMP_PLACES={88},{92},... and CUDA_VISIBLE_DEVICES=2 3

If you don't specify -N<nodes>, it will spread things across your whole allocation, unlike the default behavior for jsrun:

sierra4368{gyllen}12: **lrun -p6 js_task_info l sort**

You can specify -T <tasks_per_nodes> instead of -p<tasks>:

sierra4368{gyllen}13: **lrun -N2 -T4 js_task_info l sort**

How to see which compute nodes you were allocated

See what compute nodes you were actually allocated using lrun -T1 :

sierra4368{gyllen}5:**lrun -T1 hostname l sort**
sierra361
sierra362
<snip>

NOTE: To ssh to the first backend node, use 'lexec'. Sshing directly does not set up your environment properly for running lrun or jsrun.

Documentation for using jsrun directly (multiple threads per core examples)

There is jsrun documentation at ORNL that is quite useful: **https://www.olcf.ornl.gov/for-users/system-user-guides/summit/running-jobs/**

Here are three commonly requested scenarios for using jsrun and bsub for testing the performance of various thread layouts on a node. Below is the high-performance way to ask for a single node with either 4, 2, or 1 hardware thread per core (the smt4,smt2, or smt1 in the sbub line). The cpublink option will make the allocation take up to a minute longer to start but you will get more reproducable performance. . Run your app where js_task_info is shown (which shows your binding). All experiments are doing 1 MPI task per GPU (4 tasks per node). Hopefully with the ORNL documentation (or man jsrun), this will get you started.

jsrun with 40 OpenMP threads per MPI task, 4 tasks per node

rzansel61{gyllen}6: **bsub -nnodes 1 -core_isolation 2 -alloc_flags "smt4 cpublink autonumaoff" -Is -XF -W 60 -G guests -q pdebug /bin/tcsh**
Job <16813> is submitted to queue <pdebug>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on rzansel62>>

rzansel62{gyllen}2: **setenv OMP_NUM_THREADS 40**

rzansel62{gyllen}3: **jsrun -n 4 -c 10 -g 1 --bind rs --latency_priority=gpu-cpu,cpu-cpu,cpu-mem js_task_info**
Task 0 (0/4, 0/4) is bound to cpu[s] 0-39 on host rzansel17 with OMP_NUM_THREADS=40 and with OMP_PLACES={0:4},{4:4},{8:4},{12:4},{16:4},{20:4},{24:4},{28:4},{32:4},{36:4} and CUDA_VISIBLE_DEVICES=0
Task 1 (1/4, 1/4) is bound to cpu[s] 40-79 on host rzansel17 with OMP_NUM_THREADS=40 and with OMP_PLACES={40:4},{44:4},{48:4},{52:4},{56:4},{60:4},{64:4},{68:4},{72:4},{76:4} and CUDA_VISIBLE_DEVICES=1
Task 2 (2/4, 2/4) is bound to cpu[s] 88-127 on host rzansel17 with OMP_NUM_THREADS=40 and with OMP_PLACES={88:4},{92:4},{96:4},{100:4},{104:4},{108:4},{112:4},{116:4},{120:4},{124:4} and CUDA_VISIBLE_DEVICES=2
Task 3 (3/4, 3/4) is bound to cpu[s] 128-167 on host rzansel17 with OMP_NUM_THREADS=40 and with OMP_PLACES={128:4},{132:4},{136:4},{140:4},{144:4},{148:4},{152:4},{156:4},{160:4},{164:4} and CUDA_VISIBLE_DEVICES=3

jsrun with 20 OpenMP threads per MPI task, 4 tasks per node

rzansel61{gyllen}8: **bsub -nnodes 1 -core_isolation 2 -alloc_flags "smt2 cpublink autonumaoff" -Is -XF -W 60 -G guests -q pdebug /bin/tcsh**
Job <16822> is submitted to queue <pdebug>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on rzansel62>>

rzansel62{gyllen}2: **setenv OMP_NUM_THREADS 20**

rzansel62{gyllen}6: **jsrun -n 4 -c 10 -g 1 --bind rs --latency_priority=gpu-cpu,cpu-cpu,cpu-mem js_task_info**
Task 0 (0/4, 0/4) is bound to cpu[s] 0-1,4-5,8-9,12-13,16-17,20-21,24-25,28-29,32-33,36-37 on host rzansel18 with OMP_NUM_THREADS=20 and with OMP_PLACES={0:2},{4:2},{8:2},{12:2},{16:2},{20:2},{24:2},{28:2},{32:2},{36:2} and CUDA_VISIBLE_DEVICES=0
Task 1 (1/4, 1/4) is bound to cpu[s] 40-41,44-45,48-49,52-53,56-57,60-61,64-65,68-69,72-73,76-77 on host rzansel18 with OMP_NUM_THREADS=20 and with OMP_PLACES={40:2},{44:2},{48:2},{52:2},{56:2},{60:2},{64:2},{68:2},{72:2},{76:2} and CUDA_VISIBLE_DEVICES=1
Task 2 (2/4, 2/4) is bound to cpu[s] 88-89,92-93,96-97,100-101,104-105,108-109,112-113,116-117,120-121,124-125 on host rzansel18 with OMP_NUM_THREADS=20 and with OMP_PLACES={88:2},{92:2},{96:2},{100:2},{104:2},{108:2},{112:2},{116:2},{120:2},{124:2} and CUDA_VISIBLE_DEVICES=2
Task 3 (3/4, 3/4) is bound to cpu[s] 128-129,132-133,136-137,140-141,144-145,148-149,152-153,156-157,160-161,164-165 on host rzansel18 with OMP_NUM_THREADS=20 and with OMP_PLACES={128:2},{132:2},{136:2},{140:2},{144:2},{148:2},{152:2},{156:2},{160:2},{164:2} and CUDA_VISIBLE_DEVICES=3

jsrun with 10 OpenMP threads per MPI task, 4 tasks per node

rzansel61{gyllen}9: **bsub -nnodes 1 -core_isolation 2 -alloc_flags "smt1 cpublink autonumaoff" -Is -XF -W 60 -G guests -q pdebug /bin/tcsh**
Job <16824> is submitted to queue <pdebug>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on rzansel62>>

rzansel62{gyllen}2: **setenv OMP_NUM_THREADS 10**

rzansel62{gyllen}3: **jsrun -n 4 -c 10 -g 1 --bind rs --latency_priority=gpu-cpu,cpu-cpu,cpu-mem js_task_info**
Task 0 (0/4, 0/4) is bound to cpu[s] 0,4,8,12,16,20,24,28,32,36 on host rzansel42 with OMP_NUM_THREADS=10 and with OMP_PLACES={0},{4},{8},{12},{16},{20},{24},{28},{32},{36} and CUDA_VISIBLE_DEVICES=0
Task 1 (1/4, 1/4) is bound to cpu[s] 40,44,48,52,56,60,64,68,72,76 on host rzansel42 with OMP_NUM_THREADS=10 and with OMP_PLACES={40},{44},{48},{52},{56},{60},{64},{68},{72},{76} and CUDA_VISIBLE_DEVICES=1
Task 2 (2/4, 2/4) is bound to cpu[s] 88,92,96,100,104,108,112,116,120,124 on host rzansel42 with OMP_NUM_THREADS=10 and with OMP_PLACES={88},{92},{96},{100},{104},{108},{112},{116},{120},{124} and CUDA_VISIBLE_DEVICES=2
Task 3 (3/4, 3/4) is bound to cpu[s] 128,132,136,140,144,148,152,156,160,164 on host rzansel42 with OMP_NUM_THREADS=10 and with OMP_PLACES={128},{132},{136},{140},{144},{148},{152},{156},{160},{164} and CUDA_VISIBLE_DEVICES=3

How to set up passphrase-less keys on LC machines

If you don't have passphrase-less keys on LC machines, here are the minimal steps for setting them up. This can be done on sierra, lassen or any LC CZ machine (or rzansel or any LC RZ machine). These instructions assume you don't have any useful keys in ~/.ssh/authorized_keys. John Gyllenhaal can help if you get stuck while setting up ssh keys!

// Save existing .ssh files into .ssh.presierra just in case you had something useful in ~/.ssh/authorized_keys, etc.
sierra: **cp -rp ~/.ssh ~/.ssh.presierra**

// Generate new passphrase-less ssh keys with RSA and 2048 bits (hit <enter> at all the prompts)
sierra: **ssh-keygen -t rsa -b 2048**
Generating public/private rsa key pair.
Enter file in which to save the key (/g/g0/gyllen/.ssh/id_rsa):**<enter>**
Created directory '/g/g0/gyllen/.ssh'.
Enter passphrase (empty for no passphrase): **<enter>**
Enter same passphrase again: **<enter>**
Your identification has been saved in /g/g0/gyllen/.ssh/id_rsa.
Your public key has been saved in /g/g0/gyllen/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:ddZfv6ZNJ1gg0WrzA4ztQ3gG+5EqpqyIP/RDLYNwuLA gyllen@sierra4360
The key's randomart image is:
```
+---[RSA 2048]----+
|          ..     |
|        .  ...   |
|   .     Boo+ . .|
|. o .   +.&+ . ..o|
| o + . .SO =  . o|
|E . o * o + oo  .|
|   o.= +   ....+.|
|   o+ o       =..|
|  oo.. .      . .|
+----[SHA256]-----+
```

// Put id_rsa.pub contents at the end of ~/.ssh/authorized_keys, use cp if file doesn't exist so get appropriate permissions
sierra: **cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys**

// Make sure files are only writable by you (not group or other)
sierra: **ls -l ~/.ssh**
total 12
-rw------- 1 gyllen gyllen 399 Apr 10 11:27 authorized_keys
-rw------- 1 gyllen gyllen 1679 Apr 10 11:27 id_rsa
-rw------- 1 gyllen gyllen 399 Apr 10 11:27 id_rsa.pub

// Make sure .ssh directory only writable by you (not group or other writable)
sierra: **ls -ld ~/.ssh**
drwx------ 2 gyllen gyllen 4096 Apr 10 11:27 /g/g0/gyllen/.ssh/

// Make sure home directory only writable by you (not group or other writable)
sierra: **ls -ld ~**
drwxr-x--- 42 gyllen gyllen 24576 Apr 10 11:27 /g/g0/gyllen/

// Check that can now ssh into sierra with new passphrase-less keys.
sierra: **kdestroy**
sierra: **ssh sierra**
< Should not ask for password or passphrase. Contact John Gyllenhaal if this doesn't work!**>**

NOTE: The use of ssh-key agents with passphrases breaks jsrun/lrun but will pass the above test.

Using Managed Memory MPI buffers

Pass the -M "-gpu" flag to lrun or jsrun

MPI Collective Performance Tuning

MPI collective performance on sierra may be improved by using the Mellanox HCOLL and SHARP functionality, both of which are now enabled by default. Current benchmarking indicates that using HCOLL can reduce collective latency 10-50% for message sizes larger than 2KiB, while using SHARP can reduce collective latency 50-66% for message sizes up to 2 KiB. Best performance is observed when using both HCOLL and SHARP. As of Aug 2018, we believe we do the below by default for users but the mpiP info below may be useful for tuning parameters further for your application.

- To enable HCOLL functionality, pass the following flags to your jsrun command:

  -M "-mca coll_hcoll_enable 1 -mca coll_hcoll_np 0 -mca coll ^basic -mca coll ^ibm -HCOLL -FCA"

- To enable SHARP functionality, also pass the following flags to your jsrun command:

  -E HCOLL_SHARP_NP=2 -E HCOLL_ENABLE_SHARP=2

- If you wish ensure that SHARP is being used by your job, set the HCOLL_ENABLE_SHARP environment variable to 3, and your job will fail if it cannot use SHARP. Your job will generate messages similar to:

  [sierra2545:94746:43][common_sharp.c:292:comm_sharp_coll_init] SHArP: Fallback is disabled. exiting ...

- If you wish to generate SHARP log data indicating SHARP statistics and confirming that SHARP is being used, add -E SHARP_COLL_LOG_LEVEL=3. This will generate log data similar to:

  INFO job (ID: 4456568) resource request quota: (osts:64 user_data_per_ost:256 max_groups:0 max_qps:176 max_group_channels:1, num_trees:1)

To determine MPI collective message sizes used by an application, you can use the mpiP MPI profiler to get collective communicator and message size histogram data. To do this using the IBM-provided mpiP library, do the following:

- Load the mpip module with "module load mpip".
- Set the MPIP environment variable to "-y".
- Run your application with lrun-mpip instead of lrun.
- Your application should create an *.mpiP report file with an "Aggregate Collective Time" section with collective MPI Time %, Communicator size, and message size.
- Do not link with "-lmpiP" as this will link with the currently broken IBM mpiP library (as of 10/11/18).

Additional HCOLL environment variables can be found by running "/opt/mellanox/hcoll/bin/hcoll_info --all".

Additional SHARP environment variables can be found here.