Slurm Example Scripts

Timothy H. Kaiser, Ph.D.
tkaiser2@nrel.gov

Slides from:

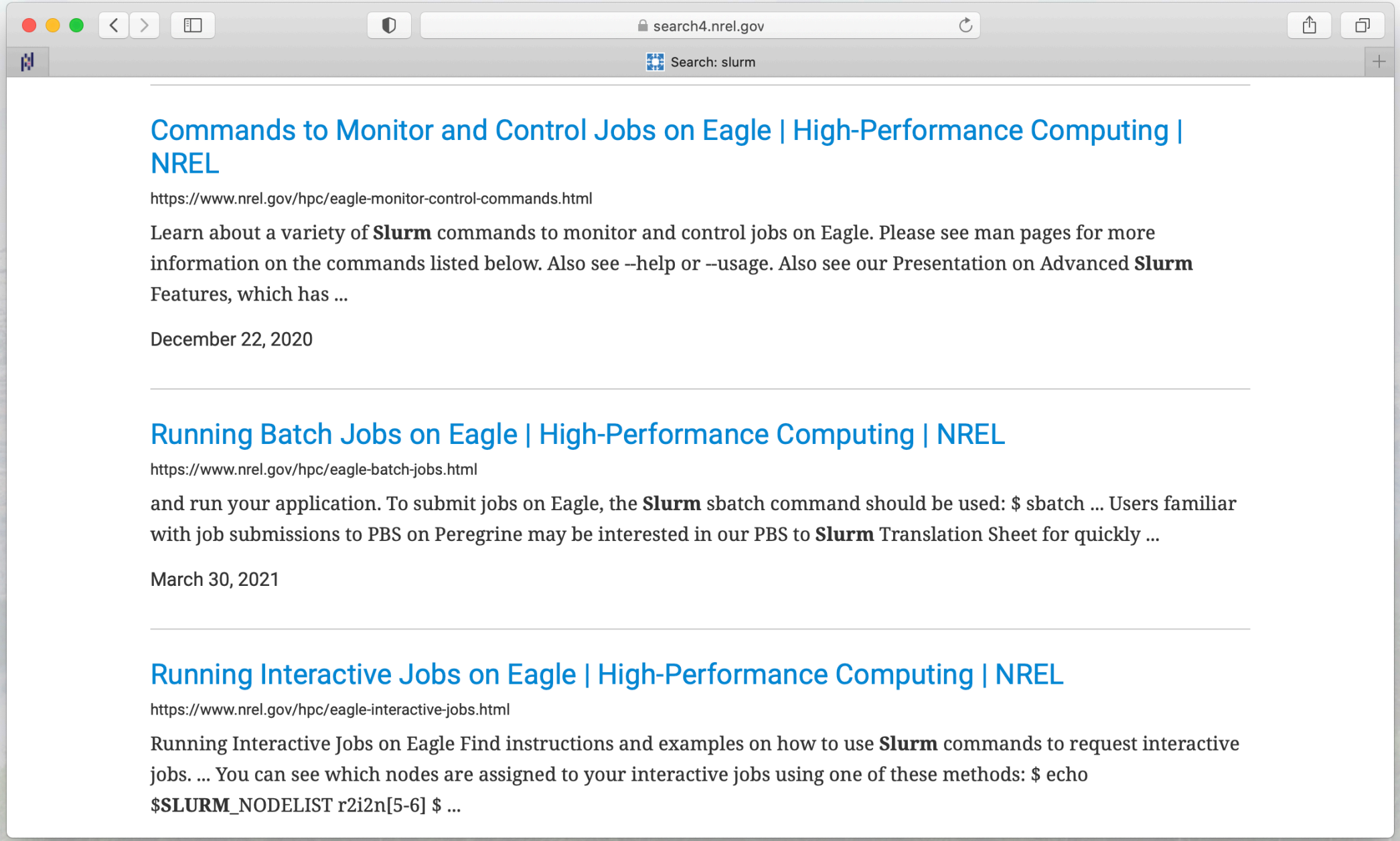
- git clone https://github.com/NREL/HPC.git
  - cd HPC/workshops/slurm_examples_slides

- https://github.com/NREL/HPC/tree/master/workshops/slurm_examples_slides

Purpose

- Generally show some techniques beyond the basics
- However, we'll briefly cover basics
- Every example is the result of a question: "How do you...?"
- Welcome more questions
- This is supplemental to NREL's HPC web pages
  - hpc.nrel.gov
  - Search for slurm

Search for slurm at hpc.nrel.gov



Slurm by example

- List of examples
- Getting them
- Building source
- Slurm commands
- Some of my custom commands
- How to run the examples
- Some important options
- The examples

Slurm by Example

- hostname.sh - Simple script that just does hostname on all cores.
- multinode-task-per-core.sh - Example of mapping process execution to each core on an arbitrary amount of nodes.
- simple.sh - Runs an MPI program with a set number of cores per node.
- hybrid.sh - Runs an MPI/OpenMP program with a set number of cores per node.
- affinity.sh - Runs an MPI/OpenMP program with a set number of cores per node, looking at various affinity settings.
- newdir.sh - Create a new directory for each run, save script, environemnt, and output

Slurm by Example

- fromenv.sh - Get input filename from your environment.
- mpmd.sh - MPI with different programs on various cores - two methods.
- mpmd2.sh - MPI with different programs on two nodes with different counts on each node.
- multi1.sh - Multiple applications in a single sbatch submission.
- gpucpu.sh - Run a cpu and gpu job in the same script concurrently.
- uselist.sh - Array jobs, multiple jobs submitted with a single script.

Slurm by Example

- FAN.sh - A bash script, not a slurm script for submitting a number of jobs with dependencies.
- CHAIN.sh - A bash script, not a slurm script for submitting a number of jobs with dependencies. A simplified version of FAN, only submits 5 jobs. See old_new.sh
- old_new_.sh - Job submitted by FAN.sh or CHAIN.sh. Can copy old run data to new directories and rerun.
- redirect.sh - Low level file redirection, allows putting slurm std{err,out} anywhere.
- multimax.sh - Multiple nodes, multiple jobs concurrently with also forcing affinity.
- local.sh - slrum script showing how to use local \"tmp\" disk.

Extras

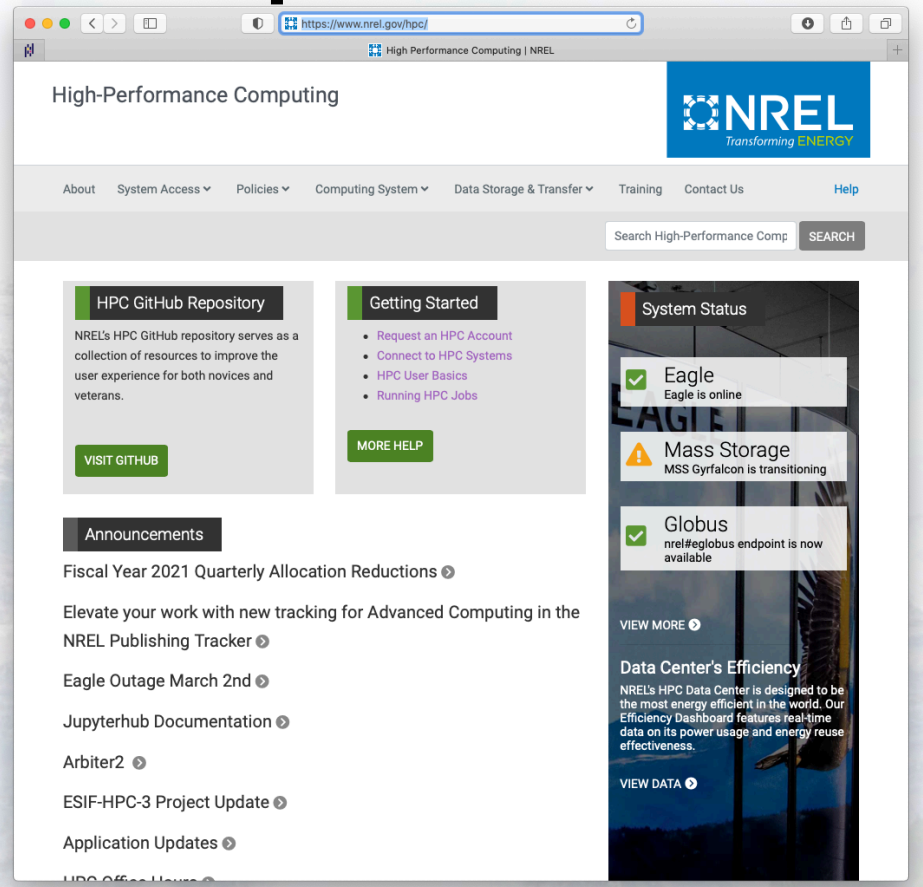
- phostone.c - Glorified hello world in hybrid MPI/OpenMP for some examples.
- doarray.py - Wrapper for uselist.sh, creates inputs and runs uselist.sh.
- stf_01.f90 - Simple finite difference code to run as an example.
- c_ex02.c - Simple example in C.
- f_ex02.f90 - Same as c_ex02.c but in Fortran.
- makefile - Makefile for examples. Loads module then compiles.
- tymer - Glorified wall clock timer.

Extras

- hymain.c - MPI program that calls a routine that uses GPUS.
- hysub.cu - Simple routine that accesses GPUs.
- invertc.c - Matrix inversion program.
- report.py - A python mpi4py program showing for mapping tasks to cores.
- spam.c - Source for a C/python library/module for mapping tasks to cores.
- setup.py - Build file for spam.c.
- jupyter.sh - Create jupyter/mpi4py/pandas environment with mpt MPI.
- tunnel.sh - Bash function for creating a ssh tunnel to connect to a jupyter notebook.
- prolog.py - A python script designed to be run as a srun prolog command.

Getting the examples

- https://www.nrel.gov/hpc/
- Click on "Visit GITHUB"
- Click on "Code"
- Copy the link



- git clone https://github.com/NREL/HPC.git
- cd HPC/slurm

Building the sources

- Go to the source directory

```
cd source
```

- Make the apps and copy them up one level up

```
make install
```

- Clean up in case you want to make with a different version of MPI

```
make clean
```

- Build a conda python environment with mpt mpi4py & jupyter

```
source jupyter.sh
```

Remove comments

- All scripts have block comments which are rather long bu informative

- You can get copies of the script without comments by running the command:

```
for script in `ls *sh` ; do
    out=`echo $script | sed s/.sh$/.slurm/`
    echo $out
    sed  '/:<<++++/,/^++++/d' $script > $out
done
```

- This is at the end of the README.md file

Some of my aliases and commands

```
alias mlist='ml 2>&1 | grep 1 | sed "s/.)//g"'
```
Nice list of loaded modules

```
alias showjob='scontrol show job'
```
Information about a job

```
alias sq='squeue -u $USER --format='\''%18A%15l%15L%6D%20S%15P%15r%20V%N'\'''
alias sqd='squeue -u $USER --format='\''%18A%15l%15L%6D%20S%15P%15r%20V%20N%20E'\'''
```
Shows information about my jobs in the queue

```
alias slist="sinfo -N -p standard -l ; sinfo -N -p gpu -l"
```
Shows show state of all nodes separated by "standard" and "gpu"

```
alias xlist='scontrol show hostname'
```
Expand a list of nodes or show nodes from inside a running job

```
alias recent='sacct -S `date --date="28 days ago" +"%Y-%m-%d"` -u $USER --format=JobID,start,elapsed,nodelist%25,state,UserCPU,WorkDir%25'
```
History of my jobs for the last 28 days

```
alias usage=/nopt/nrel/utils/bin/hours_report
```
Usage report

```
alias accounts='sacctmgr show associations user=$USER format=account%15'
```
Lists my accounts

```
alias  debug='srun --x11 --account=hpcapps --time=1:00:00 --partition=debug --ntasks=8 --cpus-per-task=4 --nodes=1 --pty bash'
alias debug1='srun --x11 --account=hpcapps --time=1:00:00 --partition=debug --ntasks=1 --cpus-per-task=8 --nodes=1 --pty bash'
alias     d2='srun --x11 --account=hpcapps --time=1:00:00 --nodes=2 --partition=debug --pty bash'
alias     d1='srun --x11 --account=hpcapps --time=1:00:00 --nodes=1 --partition=debug --pty bash'
alias     g1='srun --x11 --account=hpcapps --time=180 --ntasks=18 --nodes=1 --partition=gpu --gpus-per-node=2 --pty bash'
alias     g1d='srun --x11 --account=hpcapps --time=1:00:00 --ntasks=18 --nodes=1 --partition=debug --gpus-per-node=2 --pty bash'
alias      g2='srun --x11 --account=hpcapps --time=180 --oversubscribe --nodes=2 --partition=gpu --gpus-per-node=2 --pty bash'
alias      i1='srun --x11 --account=hpcapps --time=180 --ntasks=1 --nodes=1 --pty bash'
alias     i36='srun --x11 --account=hpcapps --time=180 --ntasks=1 --cpus-per-task=36 --nodes=1 --pty bash'
alias      i8='srun --x11 --account=hpcapps --time=180 --ntasks=1 --cpus-per-task=8 --nodes=1 --pty bash'
```
Various interactive sessions

Some of my aliases and commands

- tymer
  - Nice wall clock timer
- onodes
  - Do a "ps" on all of your nodes, showing process to core mapping
- ongpus
  - Same as above but also show gpu usage
- slurmnodes
  - Show state of nodes in a nice format
- corelist
  - List all cores from a job or list of nodes

```
git clone https://github.com/timkphd/examples
cd tims_tools
```

slurm®
workload manager

Job Submission

salloc - Obtain a job allocation.
sbatch - Submit a batch script for later execution.
srun - Obtain a job allocation (as needed) and execute an application.

--array=<indexes> (e.g. "--array=1-10")	Job array specification. (sbatch command only)
--account=<name>	Account to be charged for resources used.
--begin=<time> (e.g. "--begin=18:00:00")	Initiate job after specified time.
--clusters=<name>	Cluster(s) to run the job. (sbatch command only)
--constraint=<features>	Required node features.
--cpu_per_task=<count>	Number of CPUs required per task.
--dependency=<state:jobid>	Defer job until specified jobs reach specified state.
--error=<filename>	File in which to store job error messages.
--exclude=<names>	Specific host names to exclude from job allocation.
--exclusive[=user]	Allocated nodes can not be shared with other jobs/users.
--export=<name[=value]>	Export identified environment variables.
--gres=<name[:count]>	Generic resources required per node.
--input=<name>	File from which to read job input data.
--job-name=<name>	Job name.
--label	Prepend task ID to output. (srun command only)
--licenses=<name[:count]>	License resources required for entire job.
--mem=<MB>	Memory required per node.
--mem_per_cpu=<MB>	Memory required per allocated CPU.
-N<minnodes[-maxnodes]>	Node count required for the job.
-n<count>	Number of tasks to be launched.
--nodelist=<names>	Specific host names to include in job allocation.
--output=<name>	File in which to store job output.
--partition=<names>	Partition/queue in which to run the job.
--qos=<name>	Quality Of Service.
--signal=[B:]<num>[@time]	Signal job when approaching time limit.
--time=<time>	Wall clock time limit.
--wrap=<command_string>	Wrap specified command in a simple "sh" shell. (sbatch command only)

Accounting

sacct - Display accounting data.

--allusers	Displays all users jobs.
--accounts=<name>	Displays jobs with specified accounts.
--endtime=<time>	End of reporting period.
--format=<spec>	Format output.
--name=<jobname>	Display jobs that have any of these name(s).
--partition=<names>	Comma separated list of partitions to select jobs and job steps from.
--state=<state_list>	Display jobs with specified states.
--starttime=<time>	Start of reporting period.

SchedMD
Slurm Support and Development

sacctmgr - View and modify account information.
Options:

--immediate	Commit changes immediately.
--parseable	Output delimited by '|'

Commands:

add <ENTITY> <SPECS> create <ENTITY> <SPECS>	Add an entity. Identical to the **create** command.
delete <*ENTITY*> where <*SPECS*>	Delete the specified entities.
list <ENTITY> [<SPECS>]	Display information about the specific entity.
modify <*ENTITY*> where <*SPECS*> set <*SPECS*>	Modify an entity.

Entities:

account	Account associated with job.
cluster	*ClusterName* parameter in the *slurm.conf*.
qos	Quality of Service.
user	User name in system.

Job Management

sbcast - Transfer file to a job's compute nodes.

sbcast [options] SOURCE DESTINATION

--force	Replace previously existing file.
--preserve	Preserve modification times, access times, and access permissions.

scancel - Signal jobs, job arrays, and/or job steps.

--account=<name>	Operate only on jobs charging the specified account.
--name=<name>	Operate only on jobs with specified name.
--partition=<names>	Operate only on jobs in the specified partition/queue.
--qos=<name>	Operate only on jobs using the specified quality of service.

--reservation=<name>	Operate only on jobs using the specified reservation.
--state=<names>	Operate only on jobs in the specified state.
--user=<name>	Operate only on jobs from the specified user.
--nodelist=<names>	Operate only on jobs using the specified compute nodes.

squeue - View information about jobs.

--account=<name>	View only jobs with specified accounts.
--clusters=<name>	View jobs on specified clusters.
--format=<spec> (e.g. "--format=%i %j")	Output format to display. Specify fields, size, order, etc.
--jobs<job_id_list>	Comma separated list of job IDs to display.
--name=<name>	View only jobs with specified names.
--partition=<names>	View only jobs in specified partitions.
--priority	Sort jobs by priority.
--qos=<name>	View only jobs with specified Qualities Of Service.
--start	Report the expected start time and resources to be allocated for pending jobs in order of increasing start time.
--state=<names>	View only jobs with specified states.
--users=<names>	View only jobs for specified users.

sinfo - View information about nodes and partitions.

--all	Display information about all partitions.
--dead	If set, only report state information for non-responding (dead) nodes.
--format=<spec>	Output format to display.
--iterate=<seconds>	Print the state at specified interval.
--long	Print more detailed information.
--Node	Print information in a node-oriented format.
--partition=<names>	View only specified partitions.
--reservation	Display information about advanced reservations.
-R	Display reasons nodes are in the down, drained, fail or failing state.
--state=<names>	View only nodes specified states.

scontrol - Used view and modify configuration and state. Also see the **sview** graphical user interface version.

--details	Make show command print more details.
--oneliner	Print information on one line.

Commands:

create *SPECIFICATION*	Create a new partition or .
delete *SPECIFICATION*	Delete the entry with the specified SPECIFICATION
reconfigure	All Slurm daemons will re-read the configuration file.
requeue JOB_LIST	Requeue a running, suspended or completed batch job.
show ENTITY ID	Display the state of the specified entity with the specified identification
update *SPECIFICATION*	Update job, step, node, partition, or reservation configuration per the supplied specification.

Environment Variables

SLURM_ARRAY_JOB_ID	Set to the job ID if part of a job array.
SLURM_ARRAY_TASK_ID	Set to the task ID if part of a job array.
SLURM_CLUSTER_NAME	Name of the cluster executing the job.
SLURM_CPUS_PER_TASK	Number of CPUs requested per task.
SLURM_JOB_ACCOUNT	Account name.
SLURM_JOB_ID	Job ID.
SLURM_JOB_NAME	Job Name.
SLURM_JOB_NODELIST	Names of nodes allocated to job.
SLURM_JOB_NUM_NODES	Number of nodes allocated to job.
SLURM_JOB_PARTITION	Partition/queue running the job.
SLURM_JOB_UID	User ID of the job's owner.
SLURM_JOB_USER	User name of the job's owner.
SLURM_RESTART_COUNT	Number of times job has restarted.
SLURM_PROCID	Task ID (MPI rank).
SLURM_STEP_ID	Job step ID.
SLURM_STEP_NUM_TASKS	Task count (number of MPI ranks).

Daemons

slurmctld	Executes on cluster's "head" node to manage workload.
slurmd	Executes on each compute node to locally manage resources.
slurmdbd	Manages database of resources limits, licenses, and archives accounting records.

SchedMD
Slurm Support and Development

slurm®
workload manager

http://www.schedmd.com

Normal usage for all (most) scripts

```
sbatch –A=myaccount ––partition=debug  hostname.sh
```

Replace my account with your account.

Default Account

- You normally need to add and account to your sbatch line

```
sbatch –A=myaccount --partition=debug  hostname.sh
```

- However, you can set a default account
- Add the following to your .bashrc file replacing "hpcapps" with your account:

```
export SLURM_ACCOUNT=hpcapps
export SALLOC_ACCOUNT=hpcapps
export SBATCH_ACCOUNT=hpcapps
```

WARNING: This will have priority over what you set in a script

Some Header options

#SBATCH --account=hpcapps	Run with account “hpcapps
#SBATCH --job-name="array_job"	Show job as “array_job” in queue
#SBATCH --nodes=2	Ask for 2 nodes (see next slide)
#SBATCH --ntasks-per-node=36	36 tasks/node (see next slide)
#SBATCH --ntasks=16	16 tasks (see next slide)
#SBATCH --cpus-per-task=18	Allocate 18 cpus/task (see next slide)
#SBATCH --partition=debug	Run in the debug partition
#SBATCH --time=2-00:10:00	Maximum time is 2 days, 10 minutes
#SBATCH --error=%J.err	Errors from run will go to “job #”.err
#SBATCH --output=%J.out	Regular output from run will go to “job #”.err
#SBATCH -e stderr.%j	Same as above
#SBATCH -o stdout.%j	Same as above
#SBATCH -eo myout	Send normal and error output to the same file
#SBATCH -o /dev/null	Discard normal output
#SBATCH --gres=gpu:2	Ask for nodes with 2 GPUs
#SBATCH --mem=50000	Allocate 50,000 Mbytes on a node for this job (not normally used)
#SBATCH --exclusive	Get exclusive access to a node for this job (not normally used)
#SBATCH --exclusive=user	Get exclusive access to a node for my jobs (not normally used)

Important "count" srun options

These options can be set in your header, sbatch command line, or on the srun line.

You do not need to specify all of these but they should be consistent

```
-N, --nodes=<minnodes[-maxnodes]>
    Request that a minimum of minnodes nodes be allocated to this job.

-n, --ntasks=<number>
    Specify  the  number of tasks to run. Request that srun allocate resources for ntasks tasks.  The default
    is one task per node.

--ntasks-per-node=<ntasks>
    Request that ntasks be invoked on each node.
```

For threaded applications you should set this to be equal to the number of threads you are using OMP_NUM_THREADS

```
-c, --cpus-per-task=<ncpus>
    Request that ncpus be allocated per process. This may be useful if the job is multithreaded
    and requires more than one CPU per task for optimal performance.
```

Order of discussion

- `hostname.sh` - Simple script that just does hostname on all cores.
- `simple.sh` - Runs an MPI program with a set number of cores per node.
- `openmp.sh` - Runs an OpenMP program with a set number of cores per node.
- `hybrid.sh` - Runs an MPI/OpenMP program with a set number of cores per node.
- `affinity.sh` - Runs an MPI/OpenMP program with a set number of cores per node, looking at various affinity settings.

- `newdir.sh` - Create a new directory for each run, save script, environment, and output

- `multi.sh` - Multiple applications in a single sbatch submission.

- `mpmd.sh` - MPI with different programs on various cores - two methods.
- `mpmd2.sh` - MPI with different programs on two nodes with different counts on each node.

- `local.sh` - slurm script showing how to use local "tmp" disk.

- `fromenv.sh` - Get input filename from your environment.

- `gpucpu.sh` - Run a cpu and gpu job in the same script concurrently.

- `FAN.sh` - A bash script, not a slurm script for submitting a number of jobs with dependencies.
- `old_new_.sh` - Job submitted by FAN.sh or CHAIN.sh. Can copy old run data to new directories and rerun.

- `uselist.sh` - Array jobs, multiple jobs submitted with a single script. "Use the slurm option --array=1-24 to submit"

- `multimax.sh` - Multiple nodes, multiple jobs concurrently with also forcing affinity.
- `multinode-task-per-core.sh` - Example of mapping process execution to each core on an arbitrary amount of nodes.

- `redirect.sh` - Low level file redirection, allows putting slurm std{err,out} anywhere.

hostname.sh

Simple script that just does hostname on all cores.

```
Script that runs a serial command "hostname" on the
specified number of nodes and with the specified number of
instances.  The output will go into the slurm default output
file slurm-xxxxxx.out where xxxxxx is the job id.

Here we ask for 2 nodes and 4 tasks per node.  Specifying
ntasks=8 is redundant but if you specify all three parameters
they must be consistent.
```

hostname.sh

Simple script that just does hostname on all cores.

```
#!/bin/bash
#SBATCH --job-name="hostname"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --ntasks=8
## ask for 10 minutes
#SBATCH --time=00:10:00


# Go to the directoy from which our job was launched
cd $SLURM_SUBMIT_DIR

#run an application
srun hostname

date
```

hostname.sh

Simple script that just does hostname on all cores.

```
cat slurm-5357831.out
r105u33
r105u37
r105u33
r105u33
r105u33
r105u37
r105u37
r105u37
Thu Jan  7 10:46:36 MST 2021
```

simple.sh

Runs an MPI program with a set number of cores per node.

```
Script that runs a MPI program "purempi" on the
specified number of --nodes=2.  This can be changed
in the script or specifing a different number on the
command line.

We first run specifying the number of tasks on the srun
line.  Note that we have not specified an output file
on the command line.  In the script header we have the
line: #SBATCH --out=%J.out  This indicates that the
output will go to a file %J.out where %J is replaced
with the job number.  Any error report will go to
%J.err as specified in the header.

In our second run we do not specify the number of tasks.
It defaults to 1.

We could also set --ntasks-per-node to control the total
number of tasks to run.

We do however, pipe data into a file $SLURM_JOBID.stdout
where SLURM_JOBID is again the job number.

purempi run with the options -t 10 -T will run for 10 seconds
and print a list of nodes on which it is run and its start and
stop time.

Note the date command output will go into %J.out and %J.err should
be empty.
```

simple.sh

Runs an MPI program with a set number of tasks and node.

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err
# load our version of MPI
module load mpt

# Go to the directory from which our job was launched
cd $SLURM_SUBMIT_DIR

echo running glorified hello world

date +"starting %y-%m-%d %H:%M:%S"
echo "first run"
srun --ntasks=4  ./purempi -t 10 -T

date +"starting %y-%m-%d %H:%M:%S"
echo "second run"
srun   ./purempi -t 10 -T > $SLURM_JOBID.stdout
date +"finished %y-%m-%d %H:%M:%S"
```

We have asked for 2 nodes.

In the first case we have a total of 4 tasks which will give us 2 tasks per node.

In the second case we default to 1 task per node.

We could also set --ntasks-per-node

Command line arguments for our program

Output to this file

simple.sh

```
el1:collect> sbatch -A hpcapps --partition=debug simple.sh
Submitted batch job 5361993
el1:collect>

el1:collect> ls -l 5361993*
-rw-rw----. 1 tkaiser2 tkaiser2   0 Dec 17 10:21 5361993.err
-rw-rw----. 1 tkaiser2 tkaiser2 236 Dec 17 10:21 5361993.out
-rw-rw----. 1 tkaiser2 tkaiser2  88 Dec 17 10:21 5361993.stdout
el1:collect> cat 5361993.out
running glorified hello world
starting 20-12-17 10:21:04
first run
Thu Dec 17 10:21:05 2020
r105u37
r105u33
r105u33
r105u37
total time     10.005
Thu Dec 17 10:21:15 2020
starting 20-12-17 10:21:15
second run
finished 20-12-17 10:21:27
el1:collect>
el1:collect>

el1:collect> cat 5361993.stdout
Thu Dec 17 10:21:17 2020
r105u37
r105u33
total time     10.007
Thu Dec 17 10:21:27 2020
el1:collect>
```

Runs an MPI program with a set number of cores per node.

simple.sh

```
el1:collect> sbatch -A hpcapps --partition=debug simple.sh
Submitted batch job 5361993
el1:collect>

el1:collect> ls -l 5361993*
-rw-rw----. 1 tkaiser2 tkaiser2   0 Dec 17 10:21 5361993.err
-rw-rw----. 1 tkaiser2 tkaiser2 236 Dec 17 10:21 5361993.out
-rw-rw----. 1 tkaiser2 tkaiser2  88 Dec 17 10:21 5361993.stdout
el1:collect> cat 5361993.out
running glorified hello world
starting 20-12-17 10:21:04
first run
Thu Dec 17 10:21:05 2020
r105u37
r105u33
r105u33
r105u37
total time     10.005
Thu Dec 17 10:21:15 2020
starting 20-12-17 10:21:15
second run
finished 20-12-17 10:21:27
el1:collect>
el1:collect>

el1:collect> cat 5361993.stdout
Thu Dec 17 10:21:17 2020
r105u37
r105u33
total time     10.007
Thu Dec 17 10:21:27 2020
el1:collect>
```

Runs an MPI program with a set number of cores per node.

openmp.sh

Runs an OpenMP program with a set number of cores per node.

```
Script that runs a OpenMP program "slowinvert" on the
with a given number of OpenMP threads.

We run it with OMP_NUM_THREADS set to 18 and 36

In this version we show the effects of setting the variable
KMP_AFFINITY. KMP_AFFINITY is used to control mappings of
threads to cores when the Intel compilers are used.

The issue is that we can, if not set, see multiple threads or
tasks end up on the same core.  We will look at three settings

If

KMP_AFFINITY=verbose

a report will be sent to stderr, %J.err in our case where %J
is the job number.  The mapping of threads to cores is "default"
which is somewhat arbitrary.


KMP_AFFINITY=verbose,scatter
  and
KMP_AFFINITY=verbose,compact

We still get the report but the system tries to not map multiple
threads to the same core.

The variable KMP_AFFINITY is unique to Intel compilers.  There are
similar "OMP" variables that work for GGC compilers and Intel compilers.
For example the following settings give similar results to KMP_AFFINITY=scatter

OMP_PLACES=cores
OMP_PROC_BIND=spread
```

openmp.sh

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err
#SBATCH --cpus-per-task=36
# load our version of MPI

module load mpt
module load mkl

# Go to the directory from which our job was launched
cd $SLURM_SUBMIT_DIR

echo running a matrix inversion example with one thread per inversion

1>&2 echo  "***** running verbose,scatter *****"
export KMP_AFFINITY=verbose,scatter
export OMP_NUM_THREADS=18
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.scatter

export OMP_NUM_THREADS=36
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.scatter

unset KMP_AFFINITY
OMP_PLACES=cores
OMP_PROC_BIND=spread

export OMP_NUM_THREADS=18
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.spread

export OMP_NUM_THREADS=36
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.spread
```

Runs an OpenMP program with a set number of cores per node.

Setting KMP_AFFINITY only works if you use Intel compilers, else use the OMP__* variables

openmp.sh

Runs an OpenMP program with a set number of cores per node.

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err
#SBATCH --cpus-per-task=36
# load our version of MPI

module load mpt
module load mkl

# Go to the directory from which our job was launched
cd $SLURM_SUBMIT_DIR

echo running a matrix inversion example with one thread per inversion

1>&2 echo  "***** running verbose,scatter *****"
export KMP_AFFINITY=verbose,scatter
export OMP_NUM_THREADS=18
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.scatter

export OMP_NUM_THREADS=36
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.scatter

unset KMP_AFFINITY
OMP_PLACES=cores
OMP_PROC_BIND=spread

export OMP_NUM_THREADS=18
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.spread

export OMP_NUM_THREADS=36
#piping in /dev/null to this program gives us default input
./slowinvert < /dev/null  > $SLURM_JOBID.$OMP_NUM_THREADS.spread
```

openmp.sh

Runs an OpenMP program with a set number of cores per node.

```
sbatch -p debug -A hpcapps openmp.sh
Submitted batch job 5637141
el3:collect> ls -lt *5637141*
-rw-rw----. 1 tkaiser2 tkaiser2  7192 Jan  5 14:06 5637141.36.spread
-rw-rw----. 1 tkaiser2 tkaiser2  7192 Jan  5 14:05 5637141.18.spread
-rw-rw----. 1 tkaiser2 tkaiser2  7192 Jan  5 14:05 5637141.36.scatter
-rw-rw----. 1 tkaiser2 tkaiser2 10969 Jan  5 14:05 5637141.err
-rw-rw----. 1 tkaiser2 tkaiser2  7192 Jan  5 14:04 5637141.18.scatter
-rw-rw----. 1 tkaiser2 tkaiser2    65 Jan  5 14:03 5637141.out
el3:collect> cat 5637141.out
running a matrix inversion example with one thread per inversion

el3:collect> head 5637141.err
***** running verbose,scatter *****
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: 0-35
OMP: Info #214: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #156: KMP_AFFINITY: 36 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #285: KMP_AFFINITY: topology layer "LL cache" is equivalent to "socket".
OMP: Info #285: KMP_AFFINITY: topology layer "L3 cache" is equivalent to "socket".
OMP: Info #285: KMP_AFFINITY: topology layer "L2 cache" is equivalent to "core".
OMP: Info #285: KMP_AFFINITY: topology layer "L1 cache" is equivalent to "core".
OMP: Info #285: KMP_AFFINITY: topology layer "thread" is equivalent to "core".
el3:collect>
```

openmp.sh

Runs an OpenMP program with a set number of cores per node.

```
el3:collect> tail 5637141.18.scatter
   24    0   50665.757   50661.874       3.883
   16    0   50665.758   50661.869       3.889
   64    0   50665.786   50661.896       3.890
   72    0   50665.788   50661.897       3.891
   32    0   50665.821   50661.920       3.901
   56    0   50665.829   50661.930       3.899
   48    0   50665.832   50661.932       3.900
   40    0   50665.876   50661.984       3.892
    8    0   50665.886   50661.992       3.894
 invert time=      16.056

el3:collect> tail 5637141.36.scatter
   50    0   50701.399   50695.520       5.879
   46    0   50701.400   50695.506       5.894
   54    0   50701.412   50695.529       5.883
   22    0   50701.416   50695.516       5.900
   34    0   50701.421   50695.526       5.895
   70    0   50701.426   50695.561       5.865
   26    0   50701.426   50695.529       5.897
   58    0   50701.427   50695.533       5.894
    2    0   50701.538   50695.738       5.800
 invert time=      12.702
```

```
el3:collect> tail 5637141.18.spread
   12    0   50742.040   50738.523       3.517
   28    0   50742.096   50738.539       3.557
   64    0   50742.726   50739.334       3.392
   20    0   50743.283   50739.577       3.706
   48    0   50743.338   50739.537       3.801
    4    0   50743.416   50739.629       3.787
   16    0   50743.659   50739.923       3.736
    8    0   50743.664   50739.961       3.703
   36    0   50743.761   50739.958       3.803
 invert time=      15.974

el3:collect> tail 5637141.36.spread
   30    0   50774.025   50769.060       4.965
   42    0   50774.031   50769.106       4.925
    2    0   50774.054   50769.150       4.904
   16    0   50774.065   50769.166       4.899
   22    0   50774.074   50769.166       4.908
   28    0   50774.079   50769.178       4.901
   32    0   50774.089   50769.203       4.886
   50    0   50774.245   50769.261       4.984
   36    0   50774.345   50769.374       4.971
 invert time=      10.342
```

hybrid.sh

Runs an MPI/OpenMP program with a set number of cores per node.

```
Script that runs a hybrid MPI/OpenMP program "phostone" on the
specified number of --nodes=1 with a given number of OpenMP
threads.

Here we run with 4 tasks, 2 per node and various numbers of
threads by setting OMP_NUM_THREADS.

If we run
the program phostone with the -F option we will get a report of
mappings of MPI tasks and threads to cores.

task    thread             node name  first task    # on node  core
0000      0011               r4i7n35        0000         0000  0011
0000      0015               r4i7n35        0000         0000  0015
...
...
0001      0017               r4i7n35        0000         0001  0035

We set

OMP_PLACES=cores
OMP_PROC_BIND=spread

to ensure good mappings of threads to cores.  See the script affinity.sh
for more information on these settings.
```

hybrid.sh

Runs an MPI/OpenMP program with a set number of cores per node.

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err

# load our version of MPI
module load comp-intel
module load mpt
module load gcc

# Go to the directory from which our job was launched
cd $SLURM_SUBMIT_DIR

echo running glorified hello world

# give a good mapping of threads to cores
export OMP_PLACES=cores
export OMP_PROC_BIND=spread

for nt in 1 4 6 9 12 18 ; do
    export OMP_NUM_THREADS=$nt
    echo OMP_NUM_THREADS=$OMP_NUM_THREADS $SLURM_JOBID.$OMP_NUM_THREADS
    CORE=36
    TPN=`echo $((CORE / nt))`
    srun --nodes=2  --tasks-per-node=$TPN  --cpus-per-task=$OMP_NUM_THREADS ./phostone -t 10 -F >> $SLURM_JOBID.$OMP_NUM_THREADS
# sort the output, first grab the header
        grep core $SLURM_JOBID.$OMP_NUM_THREADS
        grep "      r" $SLURM_JOBID.$OMP_NUM_THREADS | sort -k3,3 -k6,6
        cmd "cat $SLURM_JOBID.$OMP_NUM_THREADS | awk '{print \$3, \$6}' | grep ^r | sort -u | wc " >> $SLURM_JOBID.report
done

cat $SLURM_JOBID.report
```

```
el2:collect> sbatch -A hpcapps --partition=debug hybrid.sh
Submitted batch job 633336

el2:slurm> ls -lt *633336*
-rw-rw----. 1 tkaiser2 tkaiser2  5029 Mar 23 13:18 6333364.18
-rw-rw----. 1 tkaiser2 tkaiser2 30522 Mar 23 13:18 6333364.out
-rw-rw----. 1 tkaiser2 tkaiser2   536 Mar 23 13:18 6333364.report
-rw-rw----. 1 tkaiser2 tkaiser2  5029 Mar 23 13:18 6333364.12
-rw-rw----. 1 tkaiser2 tkaiser2  5029 Mar 23 13:17 6333364.9
-rw-rw----. 1 tkaiser2 tkaiser2  5029 Mar 23 13:17 6333364.6
-rw-rw----. 1 tkaiser2 tkaiser2  5029 Mar 23 13:17 6333364.4
-rw-rw----. 1 tkaiser2 tkaiser2  5029 Mar 23 13:17 6333364.1
-rw-rw----. 1 tkaiser2 tkaiser2    87 Mar 23 13:17 6333364.err
el2:slurm>
el2:slurm> cat 6333364.1 | sort -k3,3 -k6,6 | tail -5
0029      0000              r1i7n35        0000         0029  0032
0031      0000              r1i7n35        0000         0031  0033
0033      0000              r1i7n35        0000         0033  0034
0035      0000              r1i7n35        0000         0035  0035
el2:slurm>

el2:slurm> cat 6333364.6 | sort -k3,3 -k6,6 | head -6
total time     10.006
task    thread             node name  first task    # on node  core
0006      0000               r104u33        0006         0000  0000
0006      0001               r104u33        0006         0000  0001
0006      0002               r104u33        0006         0000  0002
0006      0003               r104u33        0006         0000  0003
el2:slurm>

el2:slurm> el2:slurm> cat 6333364.6 | sort -k3,3 -k6,6 | tail -5
0005      0002              r1i7n35        0000         0005  0032
0005      0003              r1i7n35        0000         0005  0033
0005      0004              r1i7n35        0000         0005  0034
0005      0005              r1i7n35        0000         0005  0035
el2:slurm>

el2:slurm> cat 6333364.18 | sort -k3,3 -k6,6 | tail -5
0001      0014              r1i7n35        0000         0001  0032
0001      0015              r1i7n35        0000         0001  0033
0001      0016              r1i7n35        0000         0001  0034
0001      0017              r1i7n35        0000         0001  0035
el2:slurm>

el2:slurm> cat 6333364.report
+ cat 6333364.1 | awk '{print $3, $6}' | grep ^r | sort -u | wc
     72     144     936
+ cat 6333364.4 | awk '{print $3, $6}' | grep ^r | sort -u | wc
     72     144     936
+ cat 6333364.6 | awk '{print $3, $6}' | grep ^r | sort -u | wc
     72     144     936
+ cat 6333364.9 | awk '{print $3, $6}' | grep ^r | sort -u | wc
     72     144     936
+ cat 6333364.12 | awk '{print $3, $6}' | grep ^r | sort -u | wc
     72     144     936
+ cat 6333364.18 | awk '{print $3, $6}' | grep ^r | sort -u | wc
     72     144     936
el2:slurm>
```

hybrid.sh

Runs an MPI/OpenMP program with a set number of cores per node.

Runs an MPI/OpenMP program with a set number of cores per node, looking at various affinity settings.

affinity.sh

```
Script that runs a hybrid MPI/OpenMP program "phostone" on the
specified number of --nodes=1 with a given number of OpenMP
threads.

Here we run with 2 tasks and 18 threads.  We "default" to 18
threads because we have --cpus-per-task=18.  This can also be
set with the environmental variable OMP_NUM_THREADS.

In this version we show the effects of setting the variable
KMP_AFFINITY. KMP_AFFINITY is used to control mappings of
threads to cores when the Intel compilers are used.

The issue is that we can, if not set, see multiple threads or
tasks end up on the same core.  We will look at three settings

If

KMP_AFFINITY=verbose

a report will be sent to stderr, %J.err in our case where %J
is the job number.  The mapping of threads to cores is "default"
which is somewhat arbitrary.


KMP_AFFINITY=verbose,scatter
  and
KMP_AFFINITY=verbose,compact

We still get the report but the system tries to not map multiple
threads to the same core.
```

affinity.sh

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err
#SBATCH --cpus-per-task=18

# needed for threading if you use Intel compilers
module load comp-intel
# load our version of MPI
module load mpt

# Go to the directory from which our job was launched
cd $SLURM_SUBMIT_DIR

echo running glorified hello world

1>&2 echo  "***** running verbose *****"
export KMP_AFFINITY=verbose
srun --ntasks=2   ./phostone -t 10 -F > $SLURM_JOBID.noaffinity

1>&2 echo  "***** running verbose,scatter *****"
export KMP_AFFINITY=verbose,scatter
srun --ntasks=2   ./phostone -t 10 -F > $SLURM_JOBID.scatter

1>&2 echo  "***** running verbose,compact *****"
export KMP_AFFINITY=verbose,compact
srun --ntasks=2   ./phostone -t 10 -F > $SLURM_JOBID.compact

for f in noaffinity scatter compact ; do
  echo "Core report - cores over/under loaded KMP_AFFINITY=" $f
  for c in `seq -w 0 35` ; do
    echo -n "$c "
    grep 00$c\$ $SLURM_JOBID.$f | wc -l
    done | grep -v 1$
  echo " "
done
```

Runs an MPI/OpenMP program with a set number of cores per node, looking at various affinity settings.

IMPORTANT:
If you set --cups-per-task in your header it implies a setting for OMP_NUM_THREADS
The opposite is not enforced.

Runs an MPI/OpenMP program with a set number of cores per node, looking at various affinity settings.

affinity.sh

```
For each run we put the output in a seperate file.  If we run
the program phostone with the -F option we will get a report of
mappings of MPI tasks and threads to cores.

task    thread             node name  first task    # on node  core
0000      0011               r4i7n35        0000         0000  0011
0000      0015               r4i7n35        0000         0000  0015
...
...
0001      0017               r4i7n35        0000         0001  0035

We have the MPI task and OMP_THREAD_NUMBER, the node and the core number.
Ideally, for each node each core would only have a single thread.

While we can look at the individual output files to see if there is
duplication we actually we use the code in the nested for loops to
find and report where we have cores over/under loaded.  We note that
this only happend when we have KMP_AFFINITY=verbose


The variable KMP_AFFINITY is unique to Intel compilers.  There are
similar "OMP" variables that work for GGC compilers and Intel compilers.
For example the following settings give similar results to KMP_AFFINITY=scatter

OMP_PLACES=cores
OMP_PROC_BIND=spread
```

affinity.sh

Runs an MPI/OpenMP program with a set number of cores per node, looking at various affinity settings.

```
el2:collect> sbatch -A hpcapps --partition=short affinity.sh
Submitted batch job 6339780
el2:slurm>

el1:slurm> ls -lt *6339780*
-rw-rw----. 1 tkaiser2 tkaiser2   299 Mar 24 10:53 6339780.out
-rw-rw----. 1 tkaiser2 tkaiser2  2581 Mar 24 10:53 6339780.compact
-rw-rw----. 1 tkaiser2 tkaiser2 20999 Mar 24 10:53 6339780.err
-rw-rw----. 1 tkaiser2 tkaiser2  2581 Mar 24 10:53 6339780.scatter
-rw-rw----. 1 tkaiser2 tkaiser2  2581 Mar 24 10:53 6339780.noaffinity
el1:slurm> cat 6339780.out
running glorified hello world
Core report - cores over/under loaded KMP_AFFINITY= noaffinity
05 2
07 3
08 2
09 2
12 0
13 0
14 0
15 0
16 0
18 0
23 2
24 3
26 2
29 0
33 0
34 0

Core report - cores over/under loaded KMP_AFFINITY= scatter

Core report - cores over/under loaded KMP_AFFINITY= compact

el1:slurm>
```

affinity.sh

Runs an MPI/OpenMP program with a set number of cores per node, looking at various affinity settings.

```
el2:collect> head 5368560.err
running verbose
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: 0-17
OMP: Info #214: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #156: KMP_AFFINITY: 18 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #285: KMP_AFFINITY: topology layer "LL cache" is equivalent to "socket".
OMP: Info #285: KMP_AFFINITY: topology layer "L3 cache" is equivalent to "socket".
OMP: Info #285: KMP_AFFINITY: topology layer "L2 cache" is equivalent to "core".
OMP: Info #285: KMP_AFFINITY: topology layer "L1 cache" is equivalent to "core".
OMP: Info #285: KMP_AFFINITY: topology layer "thread" is equivalent to "core".
el2:collect> head 5368560.noaffinity
MPI VERSION Intel(R) MPI Library 2019 Update 7 for Linux* OS

task    thread             node name  first task    # on node  core
0000      0001               r1i1n15        0000         0000  0005
0000      0015               r1i1n15        0000         0000  0004
0000      0003               r1i1n15        0000         0000  0007
0000      0007               r1i1n15        0000         0000  0009
0000      0014               r1i1n15        0000         0000  0006
0000      0016               r1i1n15        0000         0000  0003
0000      0006               r1i1n15        0000         0000  0010
```

```
el2:collect> sbatch -A hpcapps --partition=debug hybrid.sh
Submitted batch job 5368687

el2:collect> ls -l *5368687*
-rw-rw----. 1 tkaiser2 tkaiser2  424 Dec 18 07:57 5368687.1
-rw-rw----. 1 tkaiser2 tkaiser2 1240 Dec 18 07:57 5368687.4
-rw-rw----. 1 tkaiser2 tkaiser2 2600 Dec 18 07:57 5368687.9
-rw-rw----. 1 tkaiser2 tkaiser2    0 Dec 18 07:56 5368687.err
-rw-rw----. 1 tkaiser2 tkaiser2 4126 Dec 18 07:57 5368687.out
el2:collect> cat 5368687.out
running glorified hello world
OMP_NUM_THREADS=1 5368687.1
task    thread             node name  first task    # on node  core
0000      0000               r102u34        0000         0000  0000
0001      0000               r102u34        0000         0001  0018
0002      0000               r102u35        0002         0000  0000
0003      0000               r102u35        0002         0001  0018
OMP_NUM_THREADS=4 5368687.4
task    thread             node name  first task    # on node  core
0000      0000               r102u34        0000         0000  0000
0000      0001               r102u34        0000         0000  0004
0000      0002               r102u34        0000         0000  0009
0000      0003               r102u34        0000         0000  0014
0001      0000               r102u34        0000         0001  0018
. . .
OMP_NUM_THREADS=9 5368687.9
task    thread             node name  first task    # on node  core
0000      0000               r102u34        0000         0000  0000
0000      0001               r102u34        0000         0000  0002
0000      0002               r102u34        0000         0000  0004
0000      0003               r102u34        0000         0000  0006
0000      0004               r102u34        0000         0000  0008
0000      0005               r102u34        0000         0000  0010
0000      0006               r102u34        0000         0000  0012
0000      0007               r102u34        0000         0000  0014
0000      0008               r102u34        0000         0000  0016
0001      0000               r102u34        0000         0001  0018
0001      0001               r102u34        0000         0001  0020
. . .
0003      0001               r102u35        0002         0001  0020
0003      0002               r102u35        0002         0001  0022
0003      0003               r102u35        0002         0001  0024
0003      0004               r102u35        0002         0001  0026
0003      0005               r102u35        0002         0001  0028
0003      0006               r102u35        0002         0001  0030
0003      0007               r102u35        0002         0001  0032
0003      0008               r102u35        0002         0001  0034
el2:collect>
```

affinity.sh

Runs an MPI/OpenMP program with a set number of cores per node.

Create a new directory for each run, save script, environment, and output

newdir.sh

```
This script is about creating records of what you are
doing.

It:
  creates a new directory for each run and goes there
  makes a copy of the run script
  makes a copy of the input
  saves the environment
  saves a copy of the executable
  saves a copy of stdout and stderr to the directory.

It also shows how you can use variables for your
program name ($EXE), input file ($INPUT) and argument
list ($ARGS)
```

newdir.sh

Create a new directory for each run, save script, environment, and output

```
#!/bin/bash
#SBATCH --job-name="hybrid"
#comment      = “glorified hello world"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00

# Load our version of MPI
module load mpt

# Go to the directoy from which our job was launched
cd $SLURM_SUBMIT_DIR

# Create a short JOBID base on the one provided by the scheduler
JOBID=`echo $SLURM_JOBID`

# Create a "base name" for a directory in which our job will run
# For production runs this should be in $SCRATCH
MYBASE=$SLURM_SUBMIT_DIR

# Create a directoy for our run based on the $JOBID and go there
mkdir -p $MYBASE/$JOBID
cd $MYBASE/$JOBID

# Create a link back to our starting directory
ln -s $SLURM_SUBMIT_DIR submit

# Save a copy of our script
cat $0 > script.$JOBID

# Save a copy of our environment
printenv  > env.$JOBID

# Save a list of nodes.  This is also in env.$JOBID but
# this makes it easy to find.
scontrol show hostnames > nodes.$JOBID
```

newdir.sh

Create a new directory for each run, save script, environment, and output

```
# Set the path to our program which we assume is in our starting
# directory. You should actually give the full path here. If you
# set the full path then this script is self replicating.  That
# is it can be rerun fromthe directory it creates.  An even better
# idea is to copy the executable to your new directory so we do
# that here also.

EXE=$SLURM_SUBMIT_DIR/stf_01
cp $EXE .

# Set a string which is the argument list for our program
ARGS="arg1 arg2 arg3"

# We assume that our input is in our starting directory
# We copy it here.
INPUT=st.in
cp $SLURM_SUBMIT_DIR/$INPUT .

# Run the job.
# The echo will go into the standard output for this job.  The
# standard output file will end up in the directory from which
# the job was launched.

echo "running job"
srun $EXE $ARGS < $INPUT > output.$JOBID
echo "job has finished"

# This output will also go into the standard output file
echo "run in" `pwd` " produced the files:"
ls -lt

# You can also use the following format to set
# --nodes             - # of nodes to use
# --ntasks-per-node   - ntasks = nodes*ntasks-per-node
# --ntasks            - total number of MPI tasks
#srun --nodes=$NODES --ntasks=$TASKS --ntasks-per-node=$TPN $EXE > output.$JOBID

# Copy the standard output to our run directory.

cp $SLURM_SUBMIT_DIR/slurm-$JOBID.out .
```

Create a new directory for each run, save script, environment, and output

newdir.sh

```
el2:collect> sbatch -A hpcapps --partition=debug newdir.sh
Submitted batch job 5369986
el2:collect>
el2:collect> cd 5369986
el2:5369986>
el2:5369986> ls -l
total 1032
-rw-rw----. 1 tkaiser2 tkaiser2    6365 Dec 18 10:03 env.5369986
-rw-rw----. 1 tkaiser2 tkaiser2      16 Dec 18 10:03 nodes.5369986
-rw-rw----. 1 tkaiser2 tkaiser2    1283 Dec 18 10:03 output.5369986
-rw-rw----. 1 tkaiser2 tkaiser2    6100 Dec 18 10:03 script.5369986
-rw-rw----. 1 tkaiser2 tkaiser2     565 Dec 18 10:03 slurm-5369986.out
-rwxrwx---. 1 tkaiser2 tkaiser2 1006552 Dec 18 10:03 stf_01
-rw-r-----. 1 tkaiser2 tkaiser2      54 Dec 18 10:03 st.in
lrwxrwxrwx. 1 tkaiser2 tkaiser2      22 Dec 18 10:03 submit -> /home/tkaiser2/collect
el2:5369986>
el2:5369986>
el2:5369986> head env.5369986
SLURM_NODELIST=r103u[21,23]
SLURM_JOB_NAME=hybrid
XDG_SESSION_ID=221388
SLURMD_NODENAME=r103u21
SLURM_TOPOLOGY_ADDR=root.r103.r103u21
SLURM_NTASKS_PER_NODE=8
HOSTNAME=r103u21
SPACK_ROOT=/home/tkaiser2/spack/spack
SLURM_PRIO_PROCESS=0
el2:5369986>
```

```
el2:5369986> cat nodes.5369986
r103u21
r103u23
el2:5369986>
el2:5369986> cat output.5369986
 command line argument            1 arg1
 command line argument            2 arg2
 command line argument            3 arg3
myid=    8    (  1 <= i <= 200) ,  (101 <= j <= 113)
rows=   16
myid=    0    (  1 <= i <= 200) ,  (  1 <= j <=  13)
myid=    1    (  1 <= i <= 200) ,  ( 14 <= j <=  25)
myid=    2    (  1 <= i <= 200) ,  ( 26 <= j <=  38)
myid=    3    (  1 <= i <= 200) ,  ( 39 <= j <=  50)
myid=    4    (  1 <= i <= 200) ,  ( 51 <= j <=  63)
myid=    5    (  1 <= i <= 200) ,  ( 64 <= j <=  75)
myid=    6    (  1 <= i <= 200) ,  ( 76 <= j <=  88)
myid=    7    (  1 <= i <= 200) ,  ( 89 <= j <= 100)
myid=    9    (  1 <= i <= 200) ,  (114 <= j <= 125)
myid=   10    (  1 <= i <= 200) ,  (126 <= j <= 138)
myid=   11    (  1 <= i <= 200) ,  (139 <= j <= 150)
myid=   13    (  1 <= i <= 200) ,  (164 <= j <= 175)
myid=   14    (  1 <= i <= 200) ,  (176 <= j <= 188)
myid=   15    (  1 <= i <= 200) ,  (189 <= j <= 200)
myid=   12    (  1 <= i <= 200) ,  (151 <= j <= 163)
  7500       28787803.56
 15000       1317237.087
 22500       42635.75565
 30000       1281.662011
 37500       37.85422269
 45000       1.113138052
 52500      0.3269575697E-01
 60000      0.9595943866E-03
 67500      0.1163160778E-04
 75000       0.000000000
run time =      0.88
el2:5369986>
```

newdir.sh

Create a new directory for each run, save script, environment, and output

newdir.sh

Create a new directory for each run, save script, environment, and output

```
el2:5369986>
el2:5369986> head script.5369986
#!/bin/bash
#SBATCH --job-name="hybrid"
#comment      = “glorified hello world"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00

el2:5369986>
el2:5369986> cat slurm-5369986.out
running job
job has finished
run in /home/tkaiser2/collect/5369986  produced the files:
total 1028
-rw-rw---- 1 tkaiser2 tkaiser2    1283 Dec 18 10:03 output.5369986
-rw-r----- 1 tkaiser2 tkaiser2      54 Dec 18 10:03 st.in
-rwxrwx--- 1 tkaiser2 tkaiser2 1006552 Dec 18 10:03 stf_01
-rw-rw---- 1 tkaiser2 tkaiser2      16 Dec 18 10:03 nodes.5369986
-rw-rw---- 1 tkaiser2 tkaiser2    6365 Dec 18 10:03 env.5369986
-rw-rw---- 1 tkaiser2 tkaiser2    6100 Dec 18 10:03 script.5369986
lrwxrwxrwx 1 tkaiser2 tkaiser2      22 Dec 18 10:03 submit -> /home/tkaiser2/collect
el2:5369986>
```

Multiple applications in a single sbatch submission.

multi.sh

We want to launch multiple instance of a program. First we
launch two simultaneous instances. Then we launch three
in sequence.

If we run the program phostone with the -F option we will get a report of
mappings of MPI tasks and threads to cores.

```
task    thread             node name  first task    # on node  core
0000      0011               r4i7n35        0000         0000  0011
0000      0015               r4i7n35        0000         0000  0015
...
...
0001      0017               r4i7n35        0000         0001  0035
```

With the -t option it will run for the given number of seconds.

tymer is a glorified wallclock timer. Its first argument is a file
in which to save data the rest of the arguments are comments put in
the file along with the time.

We first launch two instance of phostone and put them in the background.

We NEED the wait command. The waits for the two jobs to finish.
Without it the job may exit and the jobs could continue to run.

There is a chance that the two instance of phostone will get the same
cores. See multimax.sh for a way to force tasks to specific cores.

Finally we run the application three times in a loop with different
inputs.

Multiple applications in a single sbatch submission.

multi.sh

```
#!/bin/bash
#SBATCH --job-name="two"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00


# tymer nedas a recent version of python
module load conda

# load our version of MPI
module load mpt

export OMP_NUM_THREADS=4
#export OMP_PLACES=cores
#export OMP_PROC_BIND=spread


# Phostone.c is hello world on steroids and can be
# found at source/phostone.c.
# tymer is a glorified wallclock timer.
```

Multiple applications in a single sbatch submission.

multi.sh

```
rm -rf mytimes
# The file mytimes will have a record of what happened

# Start the first job and put it the background
./tymer mytimes start run1
srun -n 2 ./phostone -F -t 30 -T > $SLURM_JOBID.run1 &

# Start the second job and put it the background
./tymer mytimes start run2
srun -n 2 ./phostone -F -t 10 -T > $SLURM_JOBID.run2 &

# We call the wait command to wait for the jobs
# started above to finish
./tymer mytimes start waiting
wait

# Done - record our final time
./tymer mytimes done set 1

# Now we run the application three times in a loop
# with different inputs.

export OMP_PLACES=cores
export OMP_PROC_BIND=spread

for t in 5 10 15 ; do
    ./tymer mytimes doing $t
    srun -n 2 ./phostone -F -t $t -T > $SLURM_JOBID.$t
    ./tymer mytimes done $t
done
```

Multiple applications in a single sbatch submission.

multi.sh

```
el2:collect> !sbatch
sbatch -A hpcapps --partition=debug multi.sh
Submitted batch job 5380554
el2:collect> cat mytimes
1608330910.230274 Fri Dec 18 15:35:10 2020      0.000      0.000 start run1
1608330910.322142 Fri Dec 18 15:35:10 2020      0.092      0.092 start run2
1608330910.386550 Fri Dec 18 15:35:10 2020      0.064      0.156 start waiting
1608330941.063865 Fri Dec 18 15:35:41 2020     30.677     30.834 done set 1
1608330941.131300 Fri Dec 18 15:35:41 2020      0.067     30.901 doing 5
1608330947.270414 Fri Dec 18 15:35:47 2020      6.139     37.040 done 5
1608330947.333237 Fri Dec 18 15:35:47 2020      0.063     37.103 doing 10
1608330959.771684 Fri Dec 18 15:35:59 2020     12.438     49.541 done 10
1608330959.834768 Fri Dec 18 15:35:59 2020      0.063     49.604 doing 15
1608330977.523985 Fri Dec 18 15:36:17 2020     17.689     67.294 done 15
el2:collect> ls *5380554*
5380554.10  5380554.15  5380554.5  5380554.run1  5380554.run2  slurm-5380554.out
el2:collect> ls -l *5380554*
-rw-rw----. 1 tkaiser2 tkaiser2 746 Dec 18 15:35 5380554.10
-rw-rw----. 1 tkaiser2 tkaiser2 746 Dec 18 15:36 5380554.15
-rw-rw----. 1 tkaiser2 tkaiser2 746 Dec 18 15:35 5380554.5
-rw-rw----. 1 tkaiser2 tkaiser2 746 Dec 18 15:35 5380554.run1
-rw-rw----. 1 tkaiser2 tkaiser2 746 Dec 18 15:35 5380554.run2
-rw-rw----. 1 tkaiser2 tkaiser2 746 Dec 18 15:36 slurm-5380554.out
```

Multiple applications in a single sbatch submission.

multi.sh

```
el2:collect> cat 5380554.run1
Fri Dec 18 15:35:10 2020
MPI VERSION Intel(R) MPI Library 2019 Update 7 for Linux* OS

task    thread             node name  first task    # on node  core
0000      0000               r3i7n35        0000         0000  0017
0000      0003               r3i7n35        0000         0000  0000
0000      0001               r3i7n35        0000         0000  0001
0000      0002               r3i7n35        0000         0000  0002
0001      0000               r3i7n35        0000         0001  0035
0001      0002               r3i7n35        0000         0001  0020
0001      0003               r3i7n35        0000         0001  0019
0001      0001               r3i7n35        0000         0001  0021
total time     30.002
Fri Dec 18 15:35:40 2020
el2:collect> cat 5380554.run2
Fri Dec 18 15:35:10 2020
MPI VERSION Intel(R) MPI Library 2019 Update 7 for Linux* OS

task    thread             node name  first task    # on node  core
0000      0002               r3i7n35        0000         0000  0006
0000      0003               r3i7n35        0000         0000  0004
0000      0000               r3i7n35        0000         0000  0003
0000      0001               r3i7n35        0000         0000  0005
0001      0003               r3i7n35        0000         0001  0022
0001      0000               r3i7n35        0000         0001  0018
0001      0002               r3i7n35        0000         0001  0024
0001      0001               r3i7n35        0000         0001  0023
total time     10.009
Fri Dec 18 15:35:20 2020
el2:collect>
```

multinode-task-per-core.sh

Example of mapping process execution to each core on an arbitrary amount of nodes.

```
Example of mapping an echo command to each core on an
arbitrary amount of nodes using slurm. Each node on Eagle
has 36 cores, so there should be an entry for 36 * N CPU
ranks in the job output.
```

multinode-task-per-core.sh

Example of mapping process execution to each core on an arbitrary amount of nodes.

```
#!/usr/bin/env bash
#SBATCH --nodes=2   # Change this number to get different outputs
#SBATCH -t 1
#SBATCH --job-name=node_rollcall
#SBATCH -o node_rollcall.%j # output to node_rollcall.<job ID>

PROCS=$(($SLURM_NNODES * $SLURM_CPUS_ON_NODE)) # Number of CPUs * number of nodes

# Master node in jobs with N > 1 runs these
echo "I am node $SLURMD_NODENAME and I am the master node of this job with ID $SLURM_NODEID"
echo "There are $SLURM_NNODES nodes in this job, and each has $SLURM_CPUS_ON_NODE cores, for a total
of $PROCS cores."
printf "Let's get each node in the job to introduce itself:\n\n"

# Send an in-line bash script to each node to run. The single quotes prevent $var evaluation.
# `srun` uses all resources by default
srun bash <<< 'printf "\tI am $SLURMD_NODENAME, my ID for this job is $SLURM_NODEID\n"' &
wait

# Do the same, but get each node to get each core to print its "rank" (unique index)
printf "\nLet's get each node to print the ranks of all their cores (concurrently!):\n\n"
srun --ntasks=$PROCS \
     bash <<< 'printf "n${SLURM_NODEID}:c"; awk "{print \$39}" /proc/self/stat' | tr '\n' ' '
echo
```

multinode-task-per-core.sh

Example of mapping process execution to each core on an arbitrary amount of nodes.

```
I am node r5i0n13 and I am the master node of this job with ID 0
There are 2 nodes in this job, and each has 36 cores, for a total
of 72 cores.
Let's get each node in the job to introduce itself:

  I am r5i0n14, my ID for this job is 1
  I am r5i0n13, my ID for this job is 0

Let's get each node to print the ranks of all their cores
(concurrently!):

n1:c32 n0:c19 n0:c22 n1:c0 n0:c23 n1:c3 n0:c6 n1:c24 n0:c24 n1:c27
n0:c25 n1:c26 n0:c26 n1:c4 n0:c2 n1:c8 n0:c8 n1:c9 n0:c0 n1:c2
n0:c4 n1:c5 n0:c21 n1:c19 n0:c20 n1:c23 n0:c18 n1:c6 n0:c10 n1:c7
n0:c29 n0:c28 n0:c14 n0:c13 n0:c30 n0:c27 n1:c21 n0:c12 n1:c28
n0:c7 n0:c1 n0:c5 n0:c31 n1:c11 n1:c13 n1:c33 n1:c1 n1:c25 n0:c3
n1:c30 n1:c31 n1:c14 n1:c20 n1:c29 n1:c12 n1:c18 n1:c22 n1:c10
n1:c15 n0:c9 n0:c32 n0:c11 n0:c17 n0:c16 n0:c15 n1:c17 n0:c34
n1:c35 n1:c34 n1:c16 n0:c33 n0:c35
```

mpmd.sh

MPI with different programs on various cores - two methods.

```
Here we look at launching Multi Program Multi Data run.   We
do this in two ways.  We use a the --multi-prog option with
srun.  This involves creating a config_file that lists the
programs we are going to run along with the task ID.  See:

https://computing.llnl.gov/tutorials/linux_clusters/multi-prog.html

for a quick description of the format for the config_file.

Here we create the file on the fly but it could be done
beforehand.

We have two MPI programs to run together c_ex02 and f_ex02.
They are actually the same program written in C and Fortran.
But normally MPMD applications would maybe run a GUI or a
manager for  one task and rest workers.  The syntax here is

srun --multi-prog mapfile

where mapfile is the config_file.

It is possible to pass different arguments to each program
as discussed in the link above.

We also lanunch the application with mpiexec.  In this case
we just list the applications on the command line with the
number of each as would normally be done with a ":" between
listings.

These two methods both work with IntelMPI and MPT.

It is possible to do a maplist with mpiexec but that is beyound
the scope here.  Contact the author for more information.
```

mpmd.sh

MPI with different programs on various cores - two methods.

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err

# load our version of MPI
module load mpt

#create our config_file
echo "#" $SLURM_JOBID " mapfile "> mapfile
app1=./c_ex02
for n in 0 2 4 6 ; do
  echo $n $app1 >> mapfile
done
app2=./f_ex02
for n in 1 3 5 7 ; do
  echo $n $app2 >> mapfile
done

#copy it to stdout
cat mapfile

#Run with it
srun -n8 --multi-prog mapfile > used_mapfile

# The next line prevents warnings when using mpiexec with IntelMPI
unset I_MPI_PMI_LIBRARY
# Run using mpiexec 4 copies of each app
mpiexec -n 4 ./c_ex02  : -n 4 ./f_ex02 > used_mpiexec


# sort and reprint our output
echo
echo used_mapfile
grep Hello used_mapfile | sort

echo
echo used_mpiexec
grep Hello used_mpiexec | sort
```

mpmd.sh

MPI with different programs on various cores - two methods.

```
el2:collect> sbatch -A hpcapps --partition=debug mpmd.sh
Submitted batch job 5370453
el2:collect> el2:collect> ls -lt | head
total 2516
-rw-rw----. 1 tkaiser2 tkaiser2    1000 Dec 18 13:11 5370453.out
-rw-rw----. 1 tkaiser2 tkaiser2     450 Dec 18 13:11 used_mpiexec
-rw-rw----. 1 tkaiser2 tkaiser2     470 Dec 18 13:11 used_mapfile
-rw-rw----. 1 tkaiser2 tkaiser2     108 Dec 18 13:11 mapfile
-rw-rw----. 1 tkaiser2 tkaiser2       0 Dec 18 13:11 5370453.err
-rwxrwx---. 1 tkaiser2 tkaiser2   32600 Dec 18 13:10 c_ex02
-rwxrwx---. 1 tkaiser2 tkaiser2  853656 Dec 18 13:10 f_ex02
el2:collect> cat 5370453.out
# 5370453  mapfile
0 ./c_ex02
2 ./c_ex02
4 ./c_ex02
6 ./c_ex02
1 ./f_ex02
3 ./f_ex02
5 ./f_ex02
7 ./f_ex02

used_mapfile
 Hello from c process: 0  Numprocs is 8
 Hello from c process: 2  Numprocs is 8
 Hello from c process: 4  Numprocs is 8
 Hello from c process: 6  Numprocs is 8
 Hello from fortran process:            1  Numprocs is           8
 Hello from fortran process:            3  Numprocs is           8
 Hello from fortran process:            5  Numprocs is           8
 Hello from fortran process:            7  Numprocs is           8

used_mpiexec
 Hello from c process: 0  Numprocs is 8
 Hello from c process: 1  Numprocs is 8
 Hello from c process: 2  Numprocs is 8
 Hello from c process: 3  Numprocs is 8
 Hello from fortran process:            4  Numprocs is           8
 Hello from fortran process:            5  Numprocs is           8
 Hello from fortran process:            6  Numprocs is           8
 Hello from fortran process:            7  Numprocs is           8
el2:collect>
```

MPI with different programs on two nodes with different counts on each node.

mpmd.sh

```
Here we look at launching Multi Program Multi Data run.

We use a the --multi-prog option with srun.  This involves
creating a config_file that lists the programs we are going
to run along with the task ID.  See:

https://computing.llnl.gov/tutorials/linux_clusters/multi-prog.html

for a quick description of the format for the config_file.

We are using two applications c_ex02 and f_ex02.  We will be
launcing a single instance of c_ex02 and 7 copies of f_ex02.

It this case we also use hostlist which is a list of nodes
on which to run the various tasks of our simulation.

We will have c_ex02 run by itself on one node and the 7
instances of f_ex02 run on the second node.  In real life
you might want to do this, even with a single program if
task 0 will use a large amount of memory and the rest less.

The number of tasks running is fixed in this script at 8
```

MPI with different programs on two nodes with different counts on each node.

mpmd2.sh

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err

# load our version of MPI
module load mpt

#create our config_file 8 total tasks
echo "#" $SLURM_JOBID " config_file "> config_file
app1=./c_ex02
for n in 0 ; do
  echo $n $app1 >> config_file
done
app2=./f_ex02
for n in `seq 7`  ; do
  echo $n $app2 >> config_file
done

#create our hostlist file 8 total tasks
# counts should match what we have used in our config_file
counts="1 7"
# convert the counts string into an array
c_ray=(${counts})
```

MPI with different programs on two nodes with different counts on each node.

mpmd2.sh

```
# get a list of nodes
nodes=`scontrol show hostnames`
# convert the nodes string into an array
n_ray=(${nodes})
# iterate through our arrays and
#print out "count" copies of the node
len=${#n_ray[@]}
rm -rf hostlist
for (( i=0; i<$len; i++ )); do
        for x in `seq ${c_ray[$i]}` ; do
                echo "${n_ray[$i]}" >> hostlist
    done
done

echo hostlist
cat hostlist
echo

echo config_file
cat config_file

echo
export SLURM_HOSTFILE=hostlist
srun -n 8 --multi-prog config_file | sort
```

mpmd2.sh

```
el2:collect> sbatch -A hpcapps --partition=debug mpmd2.sh

el2:collect> cat 5371111.out
hostlist
r103u21
r103u23
r103u23
r103u23
r103u23
r103u23
r103u23
r103u23

config_file
# 5371111  config_file
0 ./c_ex02
1 ./f_ex02
2 ./f_ex02
3 ./f_ex02
4 ./f_ex02
5 ./f_ex02
6 ./f_ex02
7 ./f_ex02

 getting            1
Hello from c process      :    0  Numprocs is    8 r103u21
Hello from fortran process:    1  Numprocs is    8 r103u23
Hello from fortran process:    2  Numprocs is    8 r103u23
Hello from fortran process:    3  Numprocs is    8 r103u23
Hello from fortran process:    4  Numprocs is    8 r103u23
Hello from fortran process:    5  Numprocs is    8 r103u23
Hello from fortran process:    6  Numprocs is    8 r103u23
Hello from fortran process:    7  Numprocs is    8 r103u23
 i=         200
```

MPI with different programs on two nodes with different counts on each node.

slurm script showing how to use local "tmp" disk

local.sh

```
Script to show usage of local storage on Eagle.

Creates directories on local storage on each node and
puts a file with random data there.  Shows the beginning
content of the files using "od"

Uses the slurm command sbcast to copy a file to local storage.

Creates a directory in shared storage and copies the files from
local storage to the shared file system.

Deletes the local files in tmp on each node.

Note:
       2>&1 | grep -v "X11 forwarding"

    surpresses a warning from ssh.

Usage:
    sbatch -A hpcapps local.sh
```

slurm script showing how to use local "tmp" disk

local.sh

```
#!/bin/bash
#SBATCH --job-name="local"
#SBATCH --nodes=2
#SBATCH --time=00:02:00
#SBATCH --partition=debug
cd $SLURM_SUBMIT_DIR

# Create a short JOBID base on the  one provided by the scheduler
JOBID=`echo $SLURM_JOBID`

# Make a directory for this run based on the JOBID and go there
# This should be on a directory shared by all nodes, that is
# /scratch/$USER, thus we call it $SHARED

mkdir -p $JOBID
cd $JOBID
export SHARED=`pwd`

# On Eagle local disk is at /tmp/scratch
export JOBTMP=/tmp/scratch

sleep 10

#get a list of nodes...
export nlist=`scontrol show hostnames`
echo Running on: $nlist

# On the compute nodes we are going to create a local directory
# on tmp, again we use the JOBID for the name.

export MY_TMP=$JOBID
```

slurm script showing how to use local "tmp" disk

local.sh

```
echo putting data

# For each node...
for i in $nlist
do
# Create my temporary directory in /scratch on each node
  echo $i
  ssh -X $i "mkdir -p $JOBTMP/$MY_TMP" 2>&1 | grep -v "X11 forwarding"
# Create a file on each node with random data
  ssh -X  $i "head -c 1048576 </dev/urandom > $JOBTMP/$MY_TMP/afile" 2>&1 | grep -v "X11
forwarding"
# Get the first line of the file in octal
  ssh -X  $i "od $JOBTMP/$MY_TMP/afile | head -1" 2>&1 | grep -v "X11 forwarding"

done

# You can use the slurm command sbcast to put a file on each node
# here we just make a copy of your script and copy it to all nodes
cat $0 > script.$JOBID
sbcast script.$JOBID $JOBTMP/$MY_TMP/script.$JOBID
```

slurm script showing how to use local "tmp" disk

local.sh

```
echo
echo getting data
# For each node...
for i in $nlist
do
  echo $i
# Copy files from our local space on each node back to
# my working directory creating a subdirectory for each node.
# This sould be two files, our random data and a copy of our script.
  mkdir -p $SHARED/$i
  scp  -r $i:$JOBTMP/$MY_TMP/* $SHARED/$i 2>&1 | grep -v "X11 forwarding"

# Get the first line of the file in octal
  od $SHARED/$i/afile | head -1

# Remove the temporary directory
  ssh -X  $i "rm -r $JOBTMP/$MY_TMP" 2>&1 | grep -v "X11 forwarding"
done

echo
echo get listing
ls -ltR .

# Copy our output file to our new directory
cp $SLURM_SUBMIT_DIR/slurm-$JOBID.out .
```

local.sh

slurm script showing how to use local "tmp" disk

```
el3:tkaiser2> sbatch -A hpcapps local.sh
Submitted batch job 5414451
el3:tkaiser2>
el3:tkaiser2>
el3:tkaiser2>
el3:tkaiser2> ls 5414451
r105u33  r105u37  script.5414451  slurm-5414451.out
el3:tkaiser2>
el3:tkaiser2>
el3:tkaiser2> cat slurm-5414451.out
Running on: r105u33 r105u37
putting data
r105u33
0000000 054330 116353 012303 063425 164175 011147 172011 032132
r105u37
0000000 023336 021710 016745 027242 041354 055422 126172 072471

getting data
r105u33
0000000 054330 116353 012303 063425 164175 011147 172011 032132
r105u37
0000000 023336 021710 016745 027242 041354 055422 126172 072471

get listing
.:
total 9
drwxrwx--- 2 tkaiser2 tkaiser2 4096 Dec 22 11:56 r105u37
drwxrwx--- 2 tkaiser2 tkaiser2 4096 Dec 22 11:56 r105u33
-rw-rw---- 1 tkaiser2 tkaiser2 2154 Dec 22 11:56 script.5414451

./r105u37:
total 1
-rw-rw---- 1 tkaiser2 tkaiser2 1048576 Dec 22 11:56 afile
-rw-rw---- 1 tkaiser2 tkaiser2    2154 Dec 22 11:56 script.5414451

./r105u33:
total 1
-rw-rw---- 1 tkaiser2 tkaiser2 1048576 Dec 22 11:56 afile
-rw-rw---- 1 tkaiser2 tkaiser2    2154 Dec 22 11:56 script.5414451
el3:tkaiser2>
```

fromenv.sh

This script shows how to set an environmental variable before running a script and use it from within the script.

This is useful for cases where you have a bunch of runs with different input files or executables and you do not want to modify your script.

Here we set the variable INPUT before we do the sbatch command. INPUT points to our input file. In our script we check to see if it is defined and exit if not. We could also set a default value instead.

In our example output we show what happens if we first run without setting INPUT and then with it set with two different values.

Note we save a copy of our input file with the job id appended and lable our outpu with the job id also. This helps us keep track of what we did.

Get input filename from your environment.

fromenv.sh

```
#!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --out=%J.out
#SBATCH --error=%J.err


# load our version of MPI
module load mpt


if [ -z ${INPUT+x} ]; then
    echo "var is unset - quitting"
    echo "To set:"
    echo "export INPUT=afile"
    exit
else
    echo "INPUT is set to '$INPUT'"
fi

cp $INPUT $INPUT.$SLURM_JOBID
export EXE=stf_01

srun  ./$EXE  < $INPUT >> $EXE.$INPUT.output.$SLURM_JOBID
```

Get input filename from your environment.

fromenv.sh

Get input filename from your environment.

```
el2:collect> sbatch -A hpcapps --partition=debug fromenv.sh
Submitted batch job 5370160
el2:collect> ls -l *5370160*
-rw-rw----. 1 tkaiser2 tkaiser2  0 Dec 18 10:25 5370160.err
-rw-rw----. 1 tkaiser2 tkaiser2 51 Dec 18 10:25 5370160.out
el2:collect> cat 5370160.out
var is unset - quitting
To set:
export INPUT=afile
el2:collect>
el2:collect> export INPUT=st.in
el2:collect>
el2:collect> sbatch -A hpcapps --partition=debug fromenv.sh
Submitted batch job 5370166
el2:collect>
el2:collect> export INPUT=st2.in
el2:collect>
el2:collect> sbatch -A hpcapps --partition=debug fromenv.sh
Submitted batch job 5370167
el2:collect>
el2:collect> ls -l *5370166*
-rw-rw----. 1 tkaiser2 tkaiser2   0 Dec 18 10:27 5370166.err
-rw-rw----. 1 tkaiser2 tkaiser2  24 Dec 18 10:27 5370166.out
-rw-rw----. 1 tkaiser2 tkaiser2 365 Dec 18 10:28 stf_01.st.in.output.5370166
-rw-r-----. 1 tkaiser2 tkaiser2  54 Dec 18 10:27 st.in.5370166
el2:collect>
el2:collect> cat 5370166.out
INPUT is set to 'st.in'
el2:collect>
el2:collect> ls -l *5370167*
-rw-rw----. 1 tkaiser2 tkaiser2   0 Dec 18 10:28 5370167.err
-rw-rw----. 1 tkaiser2 tkaiser2  25 Dec 18 10:28 5370167.out
-rw-r-----. 1 tkaiser2 tkaiser2  54 Dec 18 10:28 st2.in.5370167
-rw-rw----. 1 tkaiser2 tkaiser2 365 Dec 18 10:29 stf_01.st2.in.output.5370167
el2:collect>
el2:collect> cat 5370167.out
INPUT is set to 'st2.in'
el2:collect>
```

Run a cpu and gpu job in the same script concurrently.

gpucpu.sh

Some workflows combine applications that use GPUs and those that only use CPUs. It is possible to run GPU applications on GPU nodes using a subset of the CPUs and simultainiously use some of the remaining CPUs to run an additional application.

The difficulty here is if you ask slurm to give you GPU access it will normally expect every application in your workflow to use them. Thus it will wait until the GPUs are free to run the CPU only applicaiton. The workaround for this situation is to unset the environmental variables while the GPUs are running.

gpucpu.sh

Run a cpu and gpu job in the same script concurrently.

```
 #!/bin/bash
#SBATCH --job-name="sample"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --gres=gpu:2
#SBATCH --out=%J.out
#SBATCH --error=%J.err



# load our version of MPI
module purge
module load mpt
module load cuda

# Make a directory for our run and go there.
mkdir $SLURM_JOB_ID
cat $0 > $SLURM_JOB_ID/script
cp input $SLURM_JOB_ID
cd $SLURM_JOB_ID

# run our MPI/GPU application but put it in the background
srun  -n 2 --gpus=2 -o mpigpu.out ../mpigpu  &
sleep 10
date

# unset these to allow slurm to schedule the CPUs
unset SLURM_JOB_GPUS
unset GPU_DEVICE_ORDINAL
unset SLURM_GPUS_PER_NODE

# run our hybrid MPI/OpenMP  application
export OMP_NUM_THREADS=6
srun --gpus=0 -o six.out -n 2 ../phostone -F -t 30

# Since we put the mpigpu application in the background we need
# to wait for it to finish before we hit the bottom of our slurm
# script or slurm could exit before it finishes.
wait
```

Run a cpu and gpu job in the same script concurrently.

gpucpu.sh

Here instead of looking at the actual program output we ssh to the node and run "ps" to show that both applications are running. We also run nvidia-smi to show the GPUs are being used.

```
el2:collect> ssh r104u33  ps -U $USER -L -o pid,lwp,psr,comm,pcpu | grep -v COMMAND | sort -k3n | egrep "phostone|mpigpu"
28907 28930   1 mpigpu           0.0
28968 29061   1 phostone         100
...
28968 29059   4 phostone         100
...
28907 28939   9 mpigpu           0.0
...
28908 28908  24 mpigpu          67.8
...
28969 28969  35 phostone         100
el2:collect> ssh r104u33 nvidia-smi
Mon Dec 21 10:51:35 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  Off  | 00000000:37:00.0 Off |                    0 |
| N/A   41C    P0    42W / 250W |    321MiB / 16160MiB |    100%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-PCIE...  Off  | 00000000:86:00.0 Off |                    0 |
| N/A   43C    P0    45W / 250W |    321MiB / 16160MiB |    100%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     28907      C   /home/tkaiser2/collect/5399183/../mpigpu     309MiB |
|    1     28908      C   /home/tkaiser2/collect/5399183/../mpigpu     309MiB |
+-----------------------------------------------------------------------------+
el2:collect>
```

FAN.sh

A bash script, not a slurm script for submitting a number of jobs with dependencies.

```
This script shows how to use slurm dependencies to build
complex workflows.

(1) Starts with 5 jobs that need to run in sequence.

(2) Submits 4 jobs that will wait for the previous 5.  However,
these 4 jobs can run at the same time.

(3) Finally a single job is run that depends or the previous 4.

The dependency graph is:

        job1
        job2
        job3
        job4
        job5
   /   /    \   \
job6 job7 job8 job9
  \   \    /    /
     job10

The way it works is that it grabs the JOBIDs returned by sbatch
and uses then as dependencies.  For jobs[2-5] this is easy since
there is a single dependency.  For jobs[6-8] we collect dependencies
in the string myset1. We note here that we could have made  jobs[6-8]
only dependent on job5 since if job5 finishes we know the others have
completed.

For job10 we collect the dependency list in the string myset2.

The slurm script we are running is old_new.sh.  It can use the
variables OLD_DIR and NEW_DIR to specify directories from which
to get data from a previous run and where to do the current run.

If OLD_DIR is defined old_new.sh will copy all files from that directory
to NEW_DIR.  There is also a variable, OLD_FILES, not used here that
can be used to only copy specific files.

For the first 5 jobs data goes into directories ser[1-5]. The next 4,
par[1-4] and the final in directory final.

Usage:

./FAN account
```

FAN.sh

A bash script, not a slurm script for submitting a number of jobs with dependencies.

```
#!/bin/bash
if [ -z ${1+x} ]; then
    echo USAGE:
    echo $0 account
    echo Your account needs to be set on the command line
    exit
fi
export ACC=$1



rm -rf ser* par* final*
echo "Starts with 5 jobs that need to run in sequence."
unset OLD_DIR
unset OLD_FILES
export NEW_DIR=ser1
jid=`sbatch --partition=short -A $ACC old_new.sh | awk '{print $NF }'`
echo $jid
myset1=""
for job in ser2 ser3 ser4 ser5 ; do
  export OLD_DIR=$NEW_DIR
  export NEW_DIR=$job
  echo --dependency=afterok:$jid
  jid=`sbatch --partition=short -A $ACC --dependency=afterok:$jid old_new.sh | awk '{print $NF }'`
  echo $jid
  myset1=$myset1,afterok:$jid
done
myset1=`echo $myset1 | sed "s/,//"`
echo $myset1

echo "Now 4 jobs that will wait for the previous 5,"
echo "however, these are independent of each other."
myset2=""
export OLD_DIR=$NEW_DIR
for job in par1 par2 par3 par4 ; do
  export NEW_DIR=$job
  echo --dependency=$myset1
  jid=`sbatch  --partition=short -A $ACC --dependency=$myset1 old_new.sh | awk '{print $NF }'`
  echo $jid
  myset2=$myset2,afterok:$jid
done
```

FAN.sh

A bash script, not a slurm script for submitting a number of jobs with dependencies.

```
echo "Finally a single job that waits for the previous 4"
getfiles=`echo $myset2 | sed "s/,/ /g"`
getfiles=`echo $getfiles | sed "s/afterok://g"`
echo $getfiles
unset OLD_DIR
export NEW_DIR=final
export OLD_FILES=$getfiles
jid=`sbatch  --partition=short -A $ACC --dependency=$myset2 old_new.sh | awk '{print $NF }'`
echo $jid

echo +-+-+- REPORT OF PENDING JOB DEPENDECIES +-+-+-
# get a list of jobs and show Dependencies
squeue  -u $LOGNAME
for jid in `squeue -h -u $LOGNAME | awk '{print $1}'` ; do
echo job $jid
scontrol show jobid -dd $jid | grep Dependency
done
```

old_new.sh

A slurm script for submitting a number of jobs with dependencies.

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:05:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j


cd $SLURM_SUBMIT_DIR
module purge
module load mpt


# Make a directory for this run and go there.
# If NEW_DIR is defined then we use that for
# our directory name or we set it to SLURM_JOBID.

if [ -z "$NEW_DIR" ]  ; then
  export NEW_DIR=$SLURM_JOBID
fi
mkdir $NEW_DIR

# If we have OLD_DIR defined then we copy old to new.
if [ -n "$OLD_DIR" ]  ; then
  cp $OLD_DIR/* $NEW_DIR
fi

# If we have OLD_FILES defined then we copy files.
# This copies a single output file from a set of
# directories instead of the whole directory.
if [ -n "$OLD_FILES" ]  ; then
  for afile in $OLD_FILES ; do
    cp */$afile.out $NEW_DIR
  done
fi

cd $NEW_DIR
export OMP_NUM_THREADS=2
# Here we just run the hello world program “phostname"
srun -n 8 $SLURM_SUBMIT_DIR/phostone -F -t 10 -T >$SLURM_JOBID.out
```

FAN.sh

A bash script, not a slurm script for submitting a number of jobs with dependencies.

```
el2:collect> ./FAN hpcapps
Start with 5 jobs that need to run in sequence.
5400356
--dependency=afterok:5400356
5400357
--dependency=afterok:5400357
5400358
--dependency=afterok:5400358
5400359
--dependency=afterok:5400359
5400360
afterok:5400357,afterok:5400358,afterok:5400359,afterok:5400360
Now 4 jobs that will wait for the previous 5,
however, these are independent of each other.
--dependency=afterok:5400357,afterok:5400358,afterok:5400359,afterok:5400360
5400361
--dependency=afterok:5400357,afterok:5400358,afterok:5400359,afterok:5400360
5400362
--dependency=afterok:5400357,afterok:5400358,afterok:5400359,afterok:5400360
5400363
--dependency=afterok:5400357,afterok:5400358,afterok:5400359,afterok:5400360
5400364
afterok:5400361,afterok:5400362,afterok:5400363,afterok:5400364
Finally a single job that waits for the previous 4
5400361 5400362 5400363 5400364
5400365
```

A bash script, not a slurm script for submitting a number of jobs with dependencies.

FAN.sh

```
+-+-+- REPORT OF PENDING JOB DEPENDECIES +-+-+-
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5400356     short    atest tkaiser2 PD       0:00      1 (None)
           5400357     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400358     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400359     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400360     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400361     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400362     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400363     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400364     short    atest tkaiser2 PD       0:00      1 (Dependency)
           5400365     short    atest tkaiser2 PD       0:00      1 (Dependency)
job 5400356
   JobState=PENDING Reason=None Dependency=(null)
job 5400357
   JobState=PENDING Reason=Dependency Dependency=afterok:5400356(unfulfilled)
job 5400358
   JobState=PENDING Reason=Dependency Dependency=afterok:5400357(unfulfilled)
job 5400359
   JobState=PENDING Reason=Dependency Dependency=afterok:5400358(unfulfilled)
job 5400360
   JobState=PENDING Reason=Dependency Dependency=afterok:5400359(unfulfilled)
job 5400361
   JobState=PENDING Reason=Dependency
Dependency=afterok:5400357(unfulfilled),afterok:5400358(unfulfilled),afterok:5400359(unfulfilled),afterok:5400360(unfulfilled)
job 5400362
   JobState=PENDING Reason=Dependency
Dependency=afterok:5400357(unfulfilled),afterok:5400358(unfulfilled),afterok:5400359(unfulfilled),afterok:5400360(unfulfilled)
job 5400363
   JobState=PENDING Reason=Dependency
Dependency=afterok:5400357(unfulfilled),afterok:5400358(unfulfilled),afterok:5400359(unfulfilled),afterok:5400360(unfulfilled)
job 5400364
   JobState=PENDING Reason=Dependency
Dependency=afterok:5400357(unfulfilled),afterok:5400358(unfulfilled),afterok:5400359(unfulfilled),afterok:5400360(unfulfilled)
job 5400365
   JobState=PENDING Reason=Dependency
Dependency=afterok:5400361(unfulfilled),afterok:5400362(unfulfilled),afterok:5400363(unfulfilled),afterok:5400364(unfulfilled)
el2:collect>
```

FAN.sh

After all jobs are done we have...

```
el2:collect> ls -d ser*
ser1  ser2  ser3  ser4  ser5
el2:collect> ls ser*
ser1:
5400356.out

ser2:
5400356.out  5400357.out

ser3:
5400356.out  5400357.out  5400358.out

ser4:
5400356.out  5400357.out  5400358.out  5400359.out

ser5:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out
el2:collect>
el2:collect>
el2:collect> ls -d par*
par1  par2  par3  par4
el2:collect> ls par*
par1:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400361.out

par2:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400362.out

par3:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400363.out

par4:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400364.out
el2:collect>
el2:collect>
el2:collect> ls -d final
final
el2:collect> ls final
5400361.out  5400362.out  5400363.out  5400364.out  5400365.out
el2:collect>
```

A bash script, not a slurm script for submitting a number of jobs with dependencies.

FAN.sh

After all jobs are done we have...

```
el2:collect> ls -d ser*
ser1  ser2  ser3  ser4  ser5
el2:collect> ls ser*
ser1:
5400356.out

ser2:
5400356.out  5400357.out

ser3:
5400356.out  5400357.out  5400358.out

ser4:
5400356.out  5400357.out  5400358.out  5400359.out

ser5:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out
el2:collect>
el2:collect>
el2:collect> ls -d par*
par1  par2  par3  par4
el2:collect> ls par*
par1:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400361.out

par2:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400362.out

par3:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400363.out

par4:
5400356.out  5400357.out  5400358.out  5400359.out  5400360.out  5400364.out
el2:collect>
el2:collect>
el2:collect> ls -d final
final
el2:collect> ls final
5400361.out  5400362.out  5400363.out  5400364.out  5400365.out
el2:collect>
```

A bash script, not a slurm script for submitting a number of jobs with dependencies.

Array jobs, multiple jobs submitted with a single script.

uselist.sh

```
This script is designed to run array jobs.  Array jobs are often
a collection of similar jobs with different inputs.


When an slurm runs a collection of array jobs it assigns two additional
variables:

SLURM_ARRAY_JOB_ID
SLURM_ARRAY_TASK_ID

Here we use SLURM_ARRAY_TASK_ID (renamed SUB_ID) to select a line from
an input file in_list.  It is expected that in_list has a line for each
SLURM_ARRAY_TASK_ID. Thus we can run N instacance of our program with
different inputs.  A typical invocation would be

sbatch -A account --array=1-24 uselist.sh

SLURM_ARRAY_TASK_ID would be in the range 1-24 and each instance would
grab one of the 24 lines of our input file.

The script creates a top level directory for all jobs and then a subdirectroy
again 1-24 for each subjob.  The subdirectory contains a copy of the input,
output, the node on which it was run, our script, the environment, and timing
information.

Here we run program that does four matrix inversions.  The matricies
are set up based on the command line input.  For example

./invertc 10 56 23 43 400

where 400 is the size and the other integers are used to set values
for the matricies.

As a convenience we have the  python script doarray.py that creates the
input file in_file and then runs this script.  doarray.py takes an account
string as input.
```

```
#!/bin/bash
#SBATCH --job-name="array_job"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=2
#SBATCH --time=00:05:00
###### sending output to /dev/null will suppress it
###### this is a good idea for array jobs lest it
###### create extra output for each subjob
#SBATCH -o /dev/null
#SBATCH --exclusive=user
#SBATCH --mem=50000

# example invocation
# sbatch -A account --array=1-24 uselist.sh

# conda is needed to get a recent version of python
module load conda

#export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS=2

#run with either the C or python version of our program
#export EXE=invertp.py
export EXE=invertc

# go to our starting directory
cd $SLURM_SUBMIT_DIR

# get the JOB and SUBJOB ID
if [[ $SLURM_ARRAY_JOB_ID ]] ; then
    export JOB_ID=$SLURM_ARRAY_JOB_ID
    export SUB_ID=$SLURM_ARRAY_TASK_ID
else
    export JOB_ID=$SLURM_JOB_ID
    export SUB_ID=1
fi
```

uselist.sh

Array jobs, multiple jobs submitted with a single script.

Array jobs are one way to "pack" a node with multiple jobs.

However, currently, Eagle will only allow a single sbatch job to run on a node. This may change in the future.

#SBATCH --exclusive=user will restrict a node to a single user. This has no effect on Eagle since it is restricted by default.

The #SBATCH --mem option sets the amount of memory each job is given; if not set a job will take all of the available memory and not allow a second job to run on the node.

Array jobs, multiple jobs submitted with a single script.

uselist.sh

```
# make a top level directory for the job
# if it does not already exist
mkdir -p $JOB_ID
cd $JOB_ID

# make a directory for the subjob and go there
mkdir -p $SUB_ID
cd $SUB_ID
# Make a copy of our script
cat $0 > myscript



# Get the name of our LIST, default to in_list
if [ -z ${LIST+x} ]; then echo "LIST is unset"; export LIST=in_list ; else echo "LIST is set to '$LIST'"; fi



# Here we assume that our each line of our LIST contains
# data for our program.
# Grab the line
    export input=`head -n $SUB_ID $SLURM_SUBMIT_DIR/$LIST | tail -1`
    printenv > envs

# Run our job
   $SLURM_SUBMIT_DIR/tymer timer start_time
   echo $input > input
   $SLURM_SUBMIT_DIR/$EXE `echo $input` > output
   hostname > node
   $SLURM_SUBMIT_DIR/tymer timer end_time
```

Array jobs, multiple jobs submitted with a single script.

uselist.sh

```
# make a top level directory for the job
# if it does not already exist
mkdir -p $JOB_ID
cd $JOB_ID

# make a directory for the subjob and go there
mkdir -p $SUB_ID
cd $SUB_ID
# Make a copy of our script
cat $0 > myscript



# Get the name of our LIST, default to in_list
if [ -z ${LIST+x} ]; then echo "LIST is unset"; export LIST=in_list ; else echo "LIST is set to '$LIST'"; fi



# Here we assume that our each line of our LIST contains
# data for our program.
# Grab the line
    export input=`head -n $SUB_ID $SLURM_SUBMIT_DIR/$LIST | tail -1`
    printenv > envs

# Run our job
   $SLURM_SUBMIT_DIR/tymer timer start_time
   echo $input > input
   $SLURM_SUBMIT_DIR/$EXE `echo $input` > output
   hostname > node
   $SLURM_SUBMIT_DIR/tymer timer end_time
```

Array jobs, multiple jobs submitted with a single script.

doarray.py

```
Set up a file, in_list, that has input for a collection of array jobs
then run them using the script uselist.sh.  Input to this script is the
account to use to run the jobs.

Example Run:

el2:array> ./doarray.py hpcapps

Example Output:

COMMAND:
sbatch -A hpcapps --array=1-24 uselist.sh
Submitted batch job 5400715

####################
We the use squeue to see what is in the queue.

el2:array> squeue -u tkaiser2
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    5400715_[1-24]     short array_jo tkaiser2 PD       0:00      1 (Priority)
el2:array>

The script uselist.sh pulls a single line from "in_list" for each instance
and uses if for input for the application it is going to run.  The
application is a matrix inversion program.  n1-n5 are used to set up
the matricies.
```

Array jobs, multiple jobs submitted with a single script.

uselist.sh

```
#!/usr/bin/python
import os
import sys
import random


if len(sys.argv[1]) < 2 :
   print("USAGE:")
   print(sys.argv[0]+" account")
   sys.exit()
account=sys.argv[1]
size=24
# make list of inputs
l=open("in_list","w")
for x in range(0,size):
   n1=int(random.random()*100)+1
   n2=int(random.random()*100)+1
   n3=int(random.random()*100)+1
   n4=int(random.random()*100)+1
   n5=400
   l.write("%d %d %d %d %d\n" % (n1,n2,n3,n4,n5))
l.close()
command="sbatch -A ACCOUNT --array=1-COUNT uselist.sh"
command=command.replace("ACCOUNT",account)
command=command.replace("COUNT",str(size))
print("COMMAND:")
print(command)
doit=os.popen(command,"r")
output=doit.readlines()
for o in output:
   print(o)
```


```
if len(sys.argv[1]) < 2 :
   print("USAGE:")
   print(sys.argv[0]+" account")
   sys.exit()
account=sys.argv[1]
size=24
# make list of inputs
l=open("in_list","w")
for x in range(0,size):
   n1=int(random.random()*100)+1
   n2=int(random.random()*100)+1
   n3=int(random.random()*100)+1
   n4=int(random.random()*100)+1
   n5=400
   l.write("%d %d %d %d %d\n" % (n1,n2,n3,n4,n5))
l.close()
command="sbatch -A ACCOUNT --array=1-COUNT uselist.sh"
command=command.replace("ACCOUNT",account)
command=command.replace("COUNT",str(size))
print("COMMAND:")
print(command)
doit=os.popen(command,"r")
output=doit.readlines()
for o in output:
   print(o)
```

Array jobs, multiple jobs submitted with a single script.

uselist.sh

```
Example output

el2:collect> ./doarray.py hpcapps
COMMAND:
sbatch -A hpcapps --array=1-24 uselist.sh
Submitted batch job 5401146

el2:collect> squeue -u tkaiser2
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    5401146_[1-24]     short array_jo tkaiser2 PD       0:00      1 (Priority)
el2:collect>
el2:collect>
el2:collect> cd 5401146
el2:5401146> ls
1  10  11  12  13  14  15  16  17  18  19  2  20  21  22  23  24  3  4  5  6  7  8  9
el2:5401146> cd 10
el2:10> ls
envs  input  myscript  node  output  timer
el2:10> cat input
7 86 11 91 400
el2:10> cat output
7 86 11 91 400
section 1 start time= 6.6042e-05   end time=    0.50803  error= 5.58091e-09
section 2 start time= 6.7949e-05   end time=    0.50685  error= 1.4797e-10
section 3 start time=    0.50725   end time=     1.0149  error= 1.91875e-09
section 4 start time=    0.50843   end time=     1.0167  error= 1.81054e-10
el2:10>
el2:10>
el2:10> cd ../22
el2:22> ls
envs  input  myscript  node  output  timer
el2:22> cat input
8 24 82 13 400
el2:22> cat output
8 24 82 13 400
section 1 start time= 6.5088e-05   end time=     0.5082  error= 3.68994e-09
section 2 start time= 6.6996e-05   end time=    0.50712  error= 8.66915e-11
section 3 start time=    0.50752   end time=     1.0152  error= 1.3155e-10
section 4 start time=     0.5086   end time=     1.0164  error= 2.47734e-09
el2:22>
```

Multiple nodes, multiple jobs concurrently with also forcing affinity.

multimax.sh

```
This script is show two ways to map MPI tasks to node when we
have multiple mpi applications running simultaneously on a
set of nodes.

The first method uses a combination of the two options: distribution
and nodelist.

From the srun man page we have:

distribution
     Specify alternate distribution methods for remote processes.

nodelist
      Request a specific list of hosts.

The second method uses the relative option.  Again from the srun
man page we have:

relative
      Run a job step relative to node n of the current allocation.

We assume we are running on two or more nodes.
```

Multiple nodes, multiple jobs concurrently with also forcing affinity.

multimax.sh

The application we are running is a python mpi4py code, report.py.
We assume we are using Intel MPI for the backend. The script jupyter.sh
can be used to create a conda environment with mpi4py with an
Intel backend in the file .

In this script our conda environement is called "dompi" as seen in
the line:

```
source activate dompi
```

Also, report.py calls two functions findcore and forcecore that are
defined in the file spam.c. After you have your conda environment
set up you can build the spam module with the command:

```
python setup.py install
```

You will want to copy it to your directory using the command:

```
cp build/lib.*/*so .
```

or set your PYTHONPATH variable to point to the directory containing
the library.

Multiple nodes, multiple jobs concurrently with also forcing affinity.

multimax.sh

```
The routines findcore and forcecore find the core on which the
calling process is running and force it to a particular core.

This is useful in this case when launcing multiple MPI runs across
node they might get mapped to the same cores.

report.py reads the environmental variable OFFSET to use as the
starting core for its layout of tasks to cores.  So we set OFFSET
before lanuching our individual copies of the program.

The first set of runs is launched in nested for loops.  We launch
a total of 4 instances with each mapped to cores[0-17] or cores[18-35]
and one of the two nodes.

In our second set of lanuches we lanuch first on the zeroth node
and then on node "one" using the relative option.
```

Multiple nodes, multiple jobs concurrently with also forcing affinity.

multimax.sh

```
#!/bin/bash
#SBATCH --job-name="mpi4py"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=36
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:01:00
#SBATCH --partition=debug
#SBATCH --account=hpcapps

# Launching multiple mpi4py programs on a collection of nodes.

# Run two sets of calculations on each one of our nodes.
# The inner loop below maps to a particular node.
# The sets are passed a command line argument 10 or 11
# from the outside loop.
# The first set runs on the first 18 cores and the second
# set runs on the last 18 cores.

# The program report.py contains code to map tasks to
# cores. It maps to core  mpi_id+OFFSET.
# For full functinality is requires installing a external
# python module.  See the source on how to do that.

# Note we are putting these MPI programs in the background
# so they run concurrently. The wait command is required.
```

Multiple nodes, multiple jobs concurrently with also forcing affinity.

multimax.sh

```
export OFFSET=0
for i in 10 11 ; do
  for n in $nodes ; do
    srun   --nodes=1 --distribution=block  -n 18  --nodelist=$n ./report.py $i > $[i]_$n &
  done
  export OFFSET=18
done
wait
date +"%y%m%d%H%M%S"
```

Multiple nodes, multiple jobs concurrently with also forcing affinity.

multimax.sh

```
export OFFSET=0
for i in 10 11 ; do
  for n in $nodes ; do
    srun   --nodes=1 --distribution=block  -n 18  --nodelist=$n ./report.py $i > $[i]_$n &
  done
  export OFFSET=18
done
wait
date +"%y%m%d%H%M%S"
```

Multiple nodes, multiple jobs concurrently with also forcing affinity.

multimax.sh

```
# Same general idea as above excpet we use the "relative" option instead of
# "distribution" and we launch in groups of 4.

date +"%y%m%d%H%M%S"
export OFFSET=0
srun   --nodes=1 --relative=0  -n 4  ./report.py 10  > run0 &
export OFFSET=4
srun   --nodes=1 --relative=0  -n 4  ./report.py 10  > run1 &
export OFFSET=0
srun   --nodes=1 --relative=1  -n 4  ./report.py 10  > run2 &
export OFFSET=4
srun   --nodes=1 --relative=1  -n 4  ./report.py 10  > run3 &
wait
date +"%y%m%d%H%M%S"

# Sort our output based on the core on which a process is running
for i in 10 11 ; do
    for n in $nodes ; do
        echo $[i]_$n
        grep Hello $[i]_$n | sort -n -k8,8
    done
done

for n in run0 run1 run2 run3 ; do
    echo $n
    grep Hello $n | sort -n -k8,8
done
```

multimax.sh

Multiple nodes, multiple jobs concurrently with also forcing affinity.

```
(/home/tkaiser2/.conda-envs/newmpi) el1> cat slurm-5306445.out
201208073517
201208073531
201208073531
201208073542
10_r103u21
xxxxxx Hello from 0 on r103u21 , 0 10
xxxxxx Hello from 1 on r103u21 , 1 10
xxxxxx Hello from 2 on r103u21 , 2 10
xxxxxx Hello from 3 on r103u21 , 3 10
xxxxxx Hello from 4 on r103u21 , 4 10
xxxxxx Hello from 5 on r103u21 , 5 10
xxxxxx Hello from 6 on r103u21 , 6 10
xxxxxx Hello from 7 on r103u21 , 7 10
xxxxxx Hello from 8 on r103u21 , 8 10
xxxxxx Hello from 9 on r103u21 , 9 10
xxxxxx Hello from 10 on r103u21 , 10 10
xxxxxx Hello from 11 on r103u21 , 11 10
xxxxxx Hello from 12 on r103u21 , 12 10
xxxxxx Hello from 13 on r103u21 , 13 10
xxxxxx Hello from 14 on r103u21 , 14 10
xxxxxx Hello from 15 on r103u21 , 15 10
xxxxxx Hello from 16 on r103u21 , 16 10
xxxxxx Hello from 17 on r103u21 , 17 10
10_r103u23
xxxxxx Hello from 0 on r103u23 , 0 10
xxxxxx Hello from 1 on r103u23 , 1 10
xxxxxx Hello from 2 on r103u23 , 2 10
xxxxxx Hello from 3 on r103u23 , 3 10
xxxxxx Hello from 4 on r103u23 , 4 10
xxxxxx Hello from 5 on r103u23 , 5 10
xxxxxx Hello from 6 on r103u23 , 6 10
xxxxxx Hello from 7 on r103u23 , 7 10
xxxxxx Hello from 8 on r103u23 , 8 10
xxxxxx Hello from 9 on r103u23 , 9 10
xxxxxx Hello from 10 on r103u23 , 10 10
xxxxxx Hello from 11 on r103u23 , 11 10
xxxxxx Hello from 12 on r103u23 , 12 10
xxxxxx Hello from 13 on r103u23 , 13 10
xxxxxx Hello from 14 on r103u23 , 14 10
xxxxxx Hello from 15 on r103u23 , 15 10
xxxxxx Hello from 16 on r103u23 , 16 10
xxxxxx Hello from 17 on r103u23 , 17 10
```

```
11_r103u21
xxxxxx Hello from 0 on r103u21 , 18 11
xxxxxx Hello from 1 on r103u21 , 19 11
xxxxxx Hello from 2 on r103u21 , 20 11
xxxxxx Hello from 3 on r103u21 , 21 11
xxxxxx Hello from 4 on r103u21 , 22 11
xxxxxx Hello from 5 on r103u21 , 23 11
xxxxxx Hello from 6 on r103u21 , 24 11
xxxxxx Hello from 7 on r103u21 , 25 11
xxxxxx Hello from 8 on r103u21 , 26 11
xxxxxx Hello from 9 on r103u21 , 27 11
xxxxxx Hello from 10 on r103u21 , 28 11
xxxxxx Hello from 11 on r103u21 , 29 11
xxxxxx Hello from 12 on r103u21 , 30 11
xxxxxx Hello from 13 on r103u21 , 31 11
xxxxxx Hello from 14 on r103u21 , 32 11
xxxxxx Hello from 15 on r103u21 , 33 11
xxxxxx Hello from 16 on r103u21 , 34 11
xxxxxx Hello from 17 on r103u21 , 35 11
11_r103u23
xxxxxx Hello from 0 on r103u23 , 18 11
xxxxxx Hello from 1 on r103u23 , 19 11
xxxxxx Hello from 2 on r103u23 , 20 11
xxxxxx Hello from 3 on r103u23 , 21 11
xxxxxx Hello from 4 on r103u23 , 22 11
xxxxxx Hello from 5 on r103u23 , 23 11
xxxxxx Hello from 6 on r103u23 , 24 11
xxxxxx Hello from 7 on r103u23 , 25 11
xxxxxx Hello from 8 on r103u23 , 26 11
xxxxxx Hello from 9 on r103u23 , 27 11
xxxxxx Hello from 10 on r103u23 , 28 11
xxxxxx Hello from 11 on r103u23 , 29 11
xxxxxx Hello from 12 on r103u23 , 30 11
xxxxxx Hello from 13 on r103u23 , 31 11
xxxxxx Hello from 14 on r103u23 , 32 11
xxxxxx Hello from 15 on r103u23 , 33 11
xxxxxx Hello from 16 on r103u23 , 34 11
xxxxxx Hello from 17 on r103u23 , 35 11
```

```
run0
xxxxxx Hello from 0 on r103u21 , 0 10
xxxxxx Hello from 1 on r103u21 , 1 10
xxxxxx Hello from 2 on r103u21 , 2 10
xxxxxx Hello from 3 on r103u21 , 3 10
run1
xxxxxx Hello from 0 on r103u21 , 4 10
xxxxxx Hello from 1 on r103u21 , 5 10
xxxxxx Hello from 2 on r103u21 , 6 10
xxxxxx Hello from 3 on r103u21 , 7 10
run2
xxxxxx Hello from 0 on r103u23 , 0 10
xxxxxx Hello from 1 on r103u23 , 1 10
xxxxxx Hello from 2 on r103u23 , 2 10
xxxxxx Hello from 3 on r103u23 , 3 10
run3
xxxxxx Hello from 0 on r103u23 , 4 10
xxxxxx Hello from 1 on r103u23 , 5 10
xxxxxx Hello from 2 on r103u23 , 6 10
xxxxxx Hello from 3 on r103u23 , 7 10
(/home/tkaiser2/.conda-envs/newmpi) el1
```

Low level file redirection, allows putting slurm std{err,out} anywhere.

redirect.sh

```
This script is rather obscure.  It again runs the hello
world example, phostone.

The purpose is to demonstrate low level redirection of
output.

Slurm has the restriction that stdout and stderr from the
script will always go to the directory from which the
script is launched.

This script shows how to get around that restriction.  In
particular, we create a file in our home directory and
stdout and stderr from slurm will go to that file.

The normal output files will still be created but they
will be empty.

This script is based on:

http://compgroups.net/comp.unix.shell/bash-changing-stdout/497180


Note that we can still use the ">" to redirect output for
individual execuatables.
```

redirect.sh

Low level file redirection, allows putting slurm std{err,out} anywhere.

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:10:00
#SBATCH --exclusive
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

cd $SLURM_SUBMIT_DIR

# Load our version of mpi
module load mpt

###############
# http://compgroups.net/comp.unix.shell/bash-changing-stdout/497180
# set up our redirects of stdout and stderr
                            # 1 and 2 are file descriptors for
                            # stdout and stderr
                            # 3 and 4 are descriptors to logfile
                            # we will use 3 for stdout 4 for stderr
exec 3>>$HOME/logfile.`date +"%y%m%d%H%M%S"`
                            # anything that goes to 4 will go to 3
                            # which is our file we have created
exec 4>&3
exec 5>&1 6>&2              # save "pointers" to stdin and stdout
exec 1>&3 2>&4              # redirect stdin and stdout to file
###############
# normal commands
                            # this line goes to stdout
echo this is a test from stdout
                            # this line goes to stderr
echo this is a test from stderr >&2
                            # error message goes to stderr
ls file_that_does_not_exist
srun -n 8 ./phostone -F -t 10 > myout.$SLURM_JOBID
srun -n 8 ./phostone -F -t 10
###############
exec 1>&5 2>&6              # restore original stdin and stdout
3>&- 4>&-                   # close logfile descriptors
5>&- 6>&-                   # close saved stdin and stdout
```

Low level file redirection, allows putting slurm std{err,out} anywhere.

redirect.sh

```
el2:collect> !sbatch
sbatch -A hpcapps --partition=debug redirect.sh
Submitted batch job 5398839
el2:collect>
el2:collect>
el2:collect> ls -lt *5398839*
-rw-rw----. 1 tkaiser2 tkaiser2 19736 Dec 21 09:07 myout.5398839
-rw-rw----. 1 tkaiser2 tkaiser2     0 Dec 21 09:07 stderr.5398839
-rw-rw----. 1 tkaiser2 tkaiser2     0 Dec 21 09:07 stdout.5398839
el2:collect>
el2:collect>
el2:collect> ls -lt ~/log*
-rw-rw----. 1 tkaiser2 tkaiser2 19940 Dec 21 09:07 /home/tkaiser2/logfile.201221090731
el2:collect>
el2:collect>
el2:collect>
el2:collect> head /home/tkaiser2/logfile.201221090731
this is a test from stdout
this is a test from stderr
ls: cannot access file_that_does_not_exist: No such file or directory
-rw-rw---- 1 tkaiser2 tkaiser2 0 Dec 21 09:07 /home/tkaiser2/collect/a_new_file
MPI VERSION Intel(R) MPI Library 2019 Update 7 for Linux* OS

task    thread             node name  first task    # on node  core
0000      0008               r2i7n35        0000         0000  0020
0000      0004               r2i7n35        0000         0000  0027
0000      0016               r2i7n35        0000         0000  0001
el2:collect>
++++
```

Jupyter Parallel Python

Timothy H. Kaiser, Ph.D.
tkaiser2@nrel.gov

What you need:

https://www.nrel.gov/hpc/system-connection.html

Windows

You will need to install an SSH client. Some free options include ~~PuTTY~~, the Windows Subsystem for Linux, **Git Bash**, and Cygwin.

For all except ~~PuTTY~~, use your SSH client to run the command:

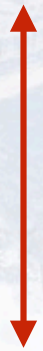
```
$ ssh username@<hostname>.hpc.nrel.gov
```

~~For PuTTY, follow the detailed instructions for using <hostname>.hpc.nrel.gov as the host name.~~

Mac or Linux

Use the built-in Terminal app to run the command:

```
$ ssh username@<hostname>.hpc.nrel.gov
```



https://gitforwindows.org

To cover:

- Build a Jupyter environment with parallel "stuff" included
- Show how to connect to Jupyter running on a compute node.
- Intro MPI & mpi4py
- Show Bag of Task with MPI
- Show Dask example

What I am using today

- git clone https://github.com/timkphd/examples
- cd examples/slurm/2020/source

Create an Environment

```
:<<++++

Author: Tim Kaiser

Build a new version of python with and Intel MPI version of mpi4py
Works with OpenMPI, just change the module load commands.


USAGE:
     source jupyter.sh

 To use the new version after the initial Install
   module load conda
   #source activate
   source activate $MYVERSION
   module load mpt

++++

### Build a new version of python with and Intel MPI version of mpi4py
CWD=`pwd`
export MYVERSION=dompi
cd ~
module load conda 2> /dev/null || echo "module load conda failed"
conda create --name $MYVERSION python=3.8 jupyter matplotlib scipy pandas xlwt dask -y

### Don't do conda init
### Just do source activate
source activate
source activate $MYVERSION

### Install mpi4py
module load mpt  2> /dev/null || echo "module load mpt failed"
pip --no-cache-dir install mpi4py


### Install slurm magic commands
pip install git+git://github.com/NERSC/slurm-magic.git

:<<++++
In a jupyter activate slurm_magix with
%load_ext slurm_magic
++++

cd $CWD
```

On Eagle, in the directory ~/examples/slurm/2020/source

Get an interactive session

```
srun --account=hpcapps --time=1:00:00 --partition=debug --nodes=1 --pty bash
```

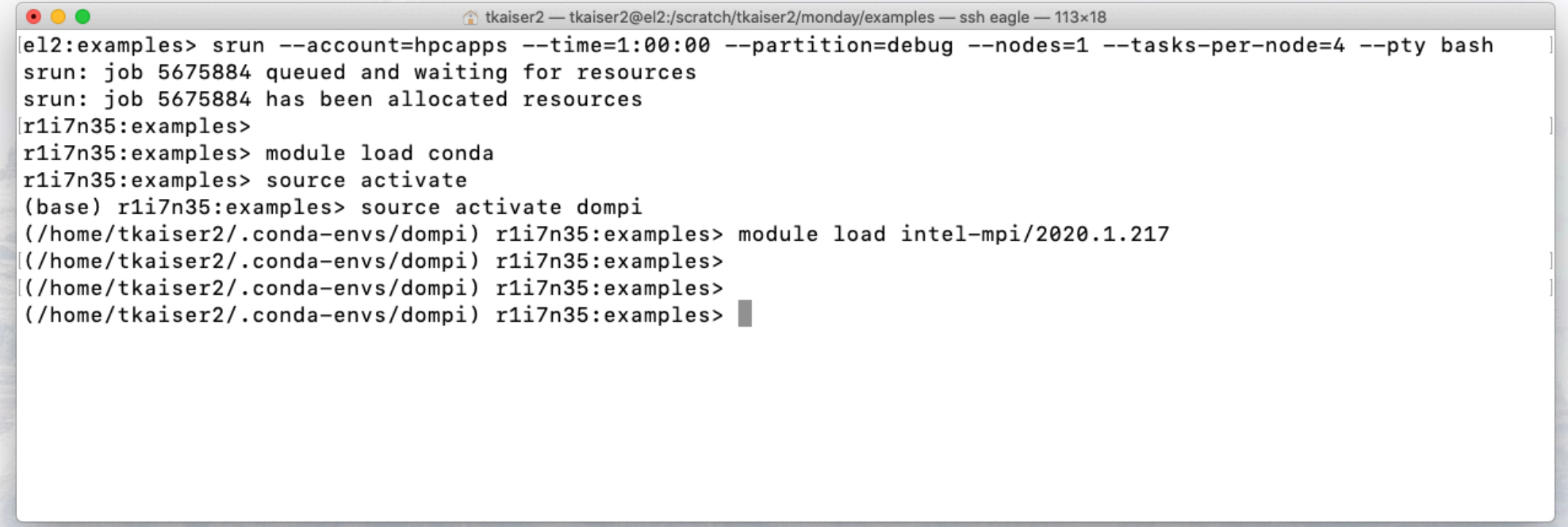
```
module load conda
source activate dompi
module load mpt
```

Start jupyter

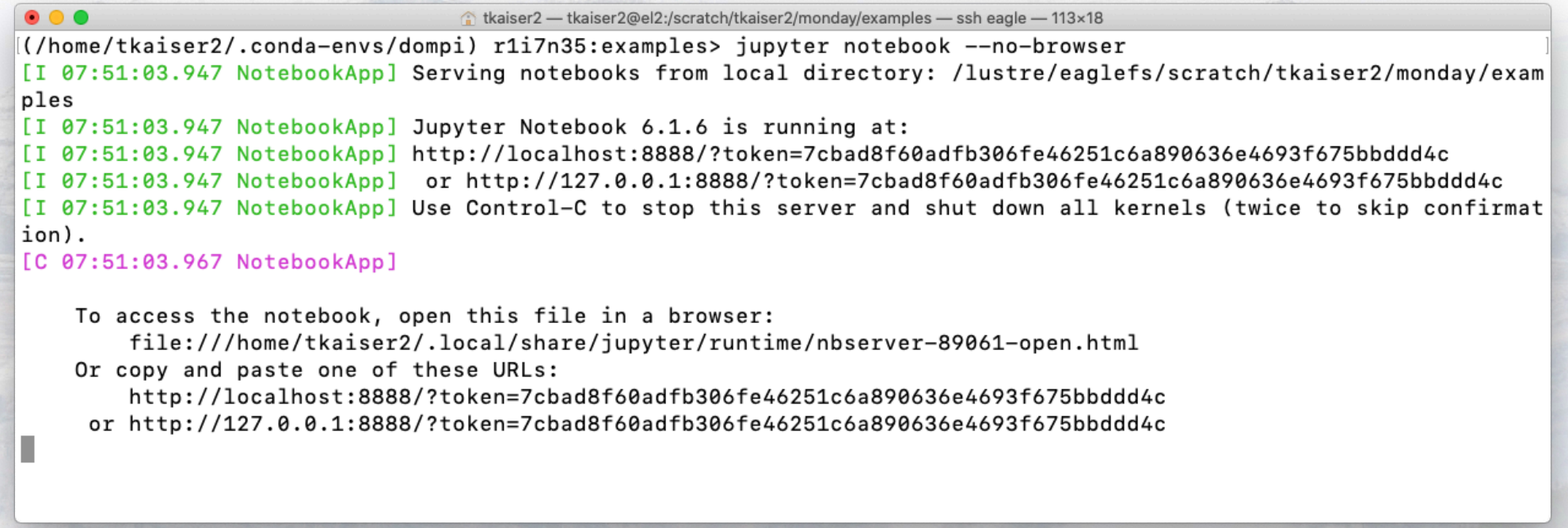
```
jupyter notebook --no-browser
```

Then the "hard" part comes...

Get a node:


tkaiser2 — tkaiser2@el2:/scratch/tkaiser2/monday/examples — ssh eagle — 113×18
el2:examples> srun --account=hpcapps --time=1:00:00 --partition=debug --nodes=1 --tasks-per-node=4 --pty bash
srun: job 5675884 queued and waiting for resources
srun: job 5675884 has been allocated resources
r1i7n35:examples>
r1i7n35:examples> module load conda
r1i7n35:examples> source activate
(base) r1i7n35:examples> source activate dompi
(/home/tkaiser2/.conda-envs/dompi) r1i7n35:examples> module load intel-mpi/2020.1.217
(/home/tkaiser2/.conda-envs/dompi) r1i7n35:examples>
(/home/tkaiser2/.conda-envs/dompi) r1i7n35:examples>
(/home/tkaiser2/.conda-envs/dompi) r1i7n35:examples>

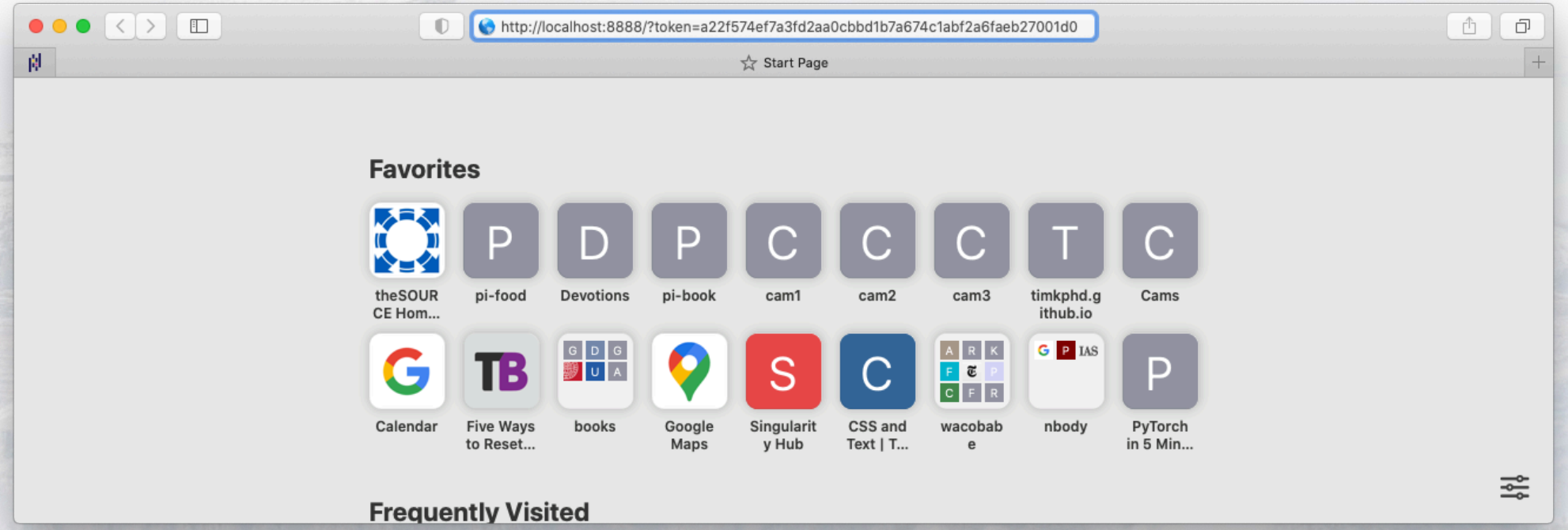
Start Jupyter



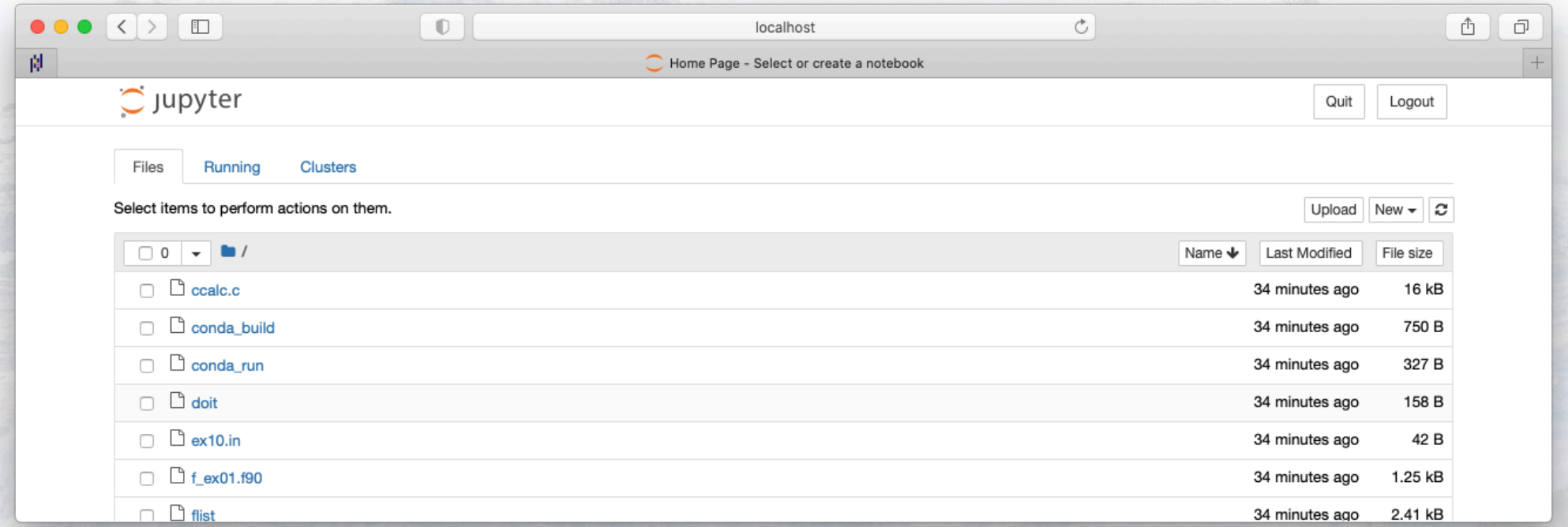
Local Machine

```
tkaiser2 — tkaiser2@r1i7n35:~ — ssh -t -L 8888:localhost:8418 eagle ssh -L 8418:localhost:8888 r1i7n35 — 89×44
mac:~> tunnel
Usage:
tunnel NODE_NAME PORT
tunnel -help
mac:~> tunnel r1i7n35 8888
Running:
ssh -t -L 8888:localhost:8418 eagle ssh -L 8418:localhost:8888 r1i7n35
********************************************************************************

                        NOTICE TO USERS

This is a Federal computer or information system and is the property of the
United States Government.  It is for authorized use only.  By using this
information system, or connecting any device to this information system, the
user acknowledges, understands and consents to certain identified actions and agrees:

  * that user has no reasonable expectation of privacy regarding communications
    or data transmitted or stored on this information system or any devices
    connected to this information system.

  * that any or all uses of this information system and all files transmitted,
    stored or connected to this information system may be intercepted, monitored,
    recorded, copied and searched and may be used or disclosed to law enforcement
    or other Government agencies, as deemed appropriate by the Department of Energy
    or as mandated by law.

  * to be bound by the requirements for use of Government information systems
    consistent with DOE Order 203.1 and any other applicable DOE Order or directive
    regarding use of DOE information systems.

  * that any unauthorized or improper use of Government information systems may
    result in limitations placed on the use of Government information systems,
    disciplinary or adverse actions, criminal penalties, and/or
    financial liability for the cost of such improper use.

To the extent that the user has any questions concerning the use of Government
information systems, the user should consult with their supervisor or other
appropriate management personnel.  LOG OFF IMMEDIATELY if you do not agree
to the conditions stated in this warning.

********************************************************************************
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Tue Dec 15 12:19:43 2020 from el1.ib0.cm.hpc.nrel.gov
r1i7n35:~>
```

Local Machine



Local Machine



To make the hard part easier...

We are going to add a function
"tunnel"
on our local machine that helps
with the connection.

```
curl \
https://raw.githubusercontent.com/timkphd/examples/master/slurm/2020/source/tunnel.sh \
| sed '/:<<++++/,/^++++/d' > tunnel.sh

cp ~/.bashrc ~/.bashrc.save

cat >> tunnel.sh ~/.bashrc
```

Next time you open a session
you'll have a new function in bash
"tunnel"

To use it:

tunnel eagle_node_name port

What it does:

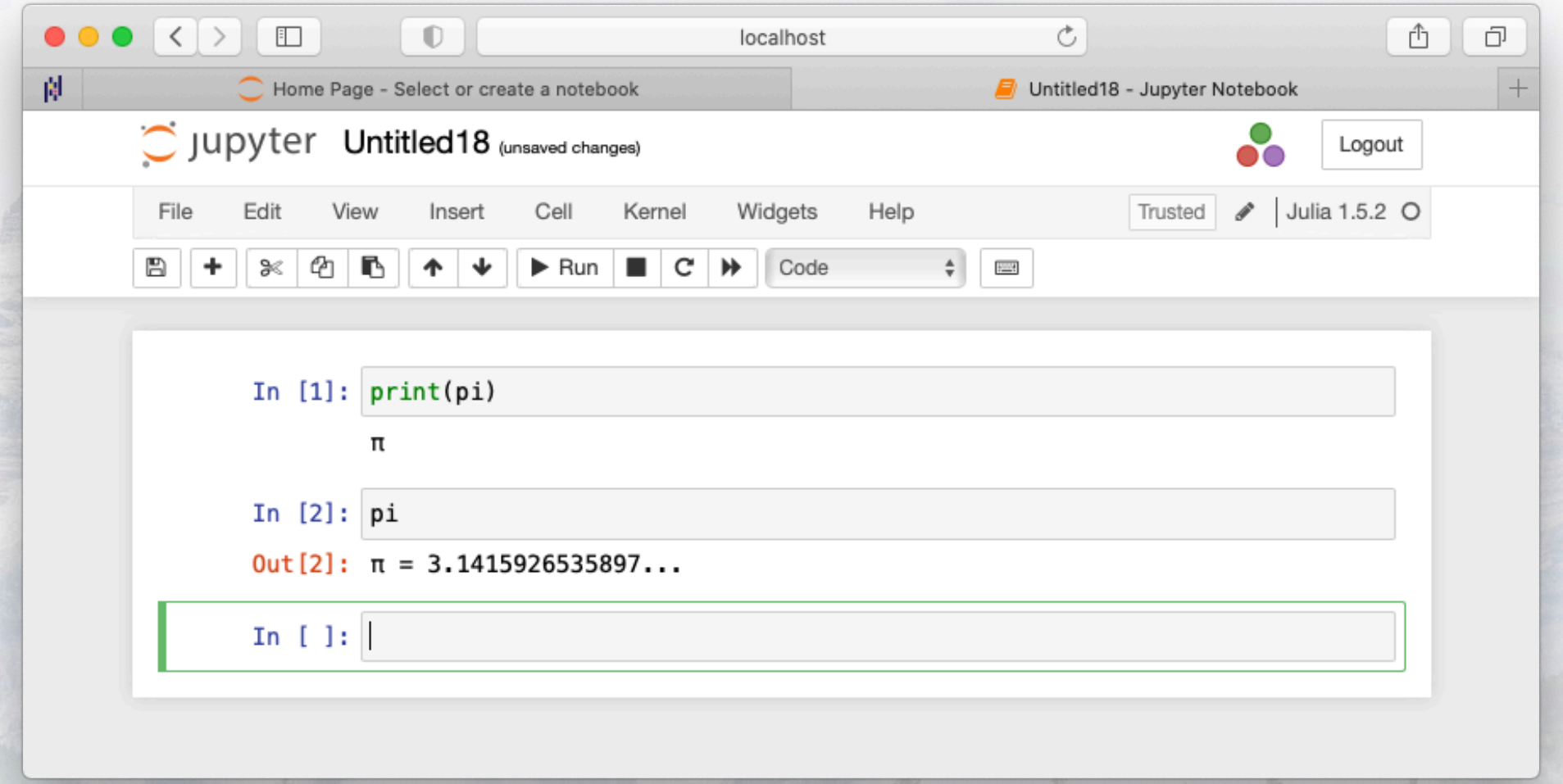
1. Connects to Eagle
2. Then to your interactive node
3. Then to your notebook

The node name and port are from the terminal window where you launched jupyter.

```
el3:~> module load julia-1.5.2-gcc-6.4.0-rkc6jax
                                                                    jupyter notebook  --NotebookApp.password='' --no-browser
el3:~> ml conda
el3:~> source activate doimpi
(/home/tkaiser2/.conda-envs/doimpi) el3:~> julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.5.2 (2020-09-23)
 _/ |\__'_|_|_|\__'_|  |
|__/                   |

julia> ]

(@v1.5) pkg>

(@v1.5) pkg> add IJulia
   Updating registry at `~/.julia/registries/General`
######################################################################## 100.0%

   Updating registry at `~/.julia/registries/JuliaComputingRegistry`
  Resolving package versions...
  Installed Parsers ──── v1.0.15
  Installed JLLWrappers ─ v1.2.0
  Installed IJulia ────── v1.23.1
Updating `~/.julia/environments/v1.5/Project.toml`
  [7073ff75] + IJulia v1.23.1
Updating `~/.julia/environments/v1.5/Manifest.toml`
  [56f22d72] + Artifacts v1.3.0
  [8f4d0f93] + Conda v1.5.0
  [7073ff75] + IJulia v1.23.1
  [692b3bcd] + JLLWrappers v1.2.0
...
...
 [8dfed614] + Test
  [cf7118a7] + UUIDs
  [4ec0a83e] + Unicode
   Building IJulia → `~/.julia/packages/IJulia/IDNmS/deps/build.log`

(@v1.5) pkg>
(@v1.5) pkg> ^C


julia>
(/home/tkaiser2/.conda-envs/doimpi) el3:~> jupyter notebook
```

https://github.com/JuliaLang/IJulia.jl

https://github.com/jupyter/jupyter/wiki/Jupyter-k


localhost
Home Page - Select or create a notebook
Untitled18 - Jupyter Notebook
jupyter Untitled18 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Julia 1.5.2
Run
Code
In [1]: print(pi)
π
In [2]: pi
Out[2]: π = 3.1415926535897...
In [ ]:

Parallel

- Regular MPI - mpi4py
- MPI - bag of tasks
- Dask

Parallel

Get bold: ask for two nodes:

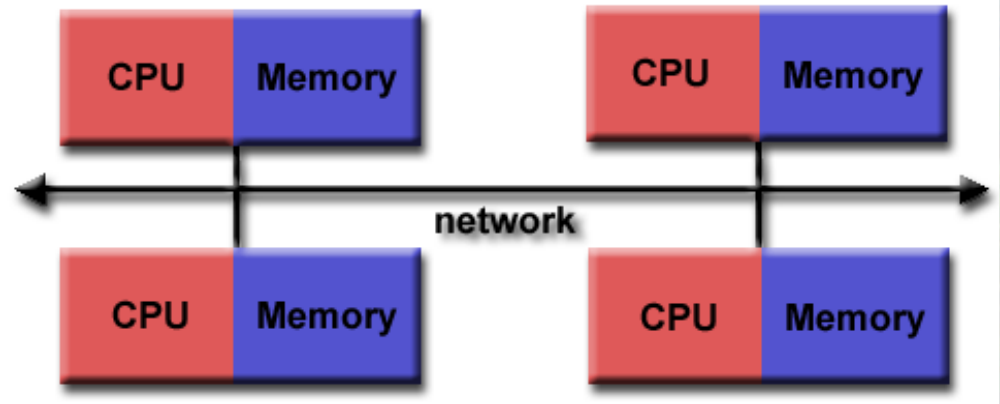
```
srun --account=hpcapps --time=1:00:00 --partition=debug --nodes=2 --tasks-per-node=36 --pty bash
```

tkaiser2 — tkaiser2@el2:/scratch/tkaiser2/monday/examples/mpi/mpi4py — ssh eagle — 112×24

```
[el2:mpi4py> srun --account=hpcapps --time=1:00:00 --partition=debug --nodes=2 --tasks-per-node=36 --pty bash ]
srun: job 5676338 queued and waiting for resources
srun: job 5676338 has been allocated resources
[r102u34:mpi4py>                                                                                              ]
[r102u34:mpi4py>                                                                                              ]
[r102u34:mpi4py>                                                                                              ]
r102u34:mpi4py> module load conda
r102u34:mpi4py> source activate
(base) r102u34:mpi4py> source activate dompi
[(/home/tkaiser2/.conda-envs/dompi) r102u34:mpi4py> module load intel-mpi/2020.1.217                          ]
(/home/tkaiser2/.conda-envs/dompi) r102u34:mpi4py>
```

MPI

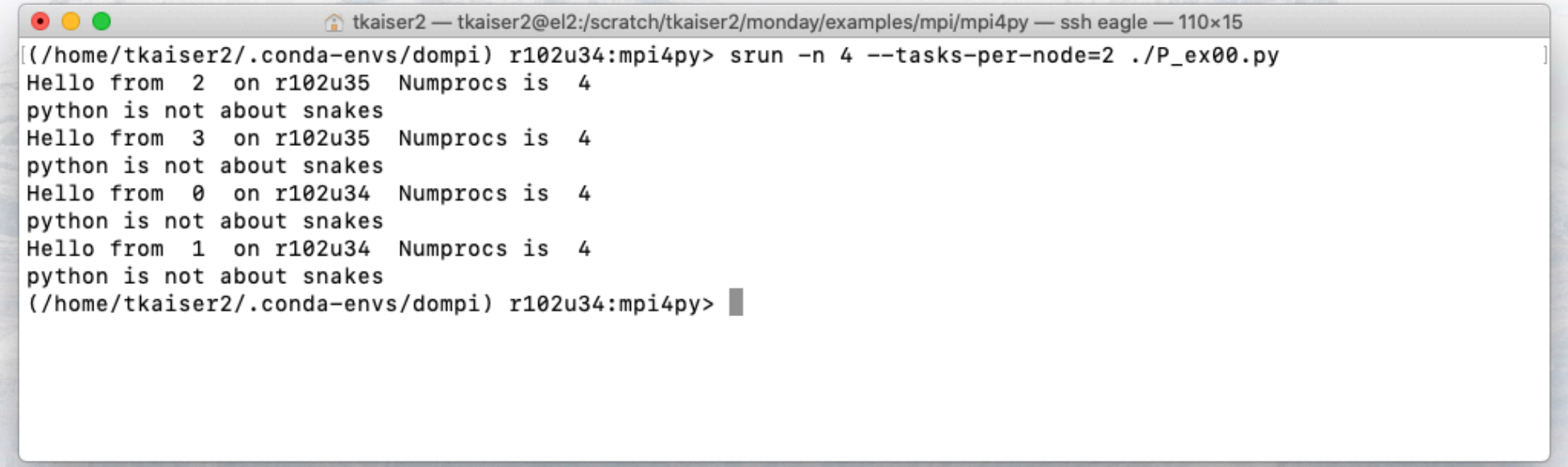
- Message Passing Interface (MPI) is a standardized and portable message-passing standard designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures.
- https://www.mpi-forum.org



Hello world in MPI

```
#!/usr/bin/env python
# numpy is required
import numpy
from numpy import *

# mpi4py module
from mpi4py import MPI

# Initialize MPI
comm=MPI.COMM_WORLD

# What processor am I (what is my rank)?
myid=comm.Get_rank()

# How many processors (nPEs) are there?
numprocs=comm.Get_size()

# Processor name?
name = MPI.Get_processor_name()

print("Hello from ",myid," on",name," Numprocs is ",numprocs)

print("python is not about snakes")

# Shut down MPI
MPI.Finalize()
```

```
srun -n 4 --tasks-per-node=2 ./P_ex00.py
```



Bag of tasks

- One of the most common things that is done in parallel python is "bag of tasks"
- Many identical tasks except different input
- The developer of MPI4PY created a framework for doing BoT.

~/examples/mpi/mpi4py/queue.py

```
notes="""
This program shows how to use mpi4py's built in bag of task feature.
The work being performed is defined in the dummy function task.  task
gets values from its argument list.  In this case we are passing it
a random word from the function ranword and an integer.  The integer
is just the index of the task that is being run.  task creates a
string out of the arguments, sleeps for some time and then returns.

We create a dictionary "future" to hold a list of running tasks.

In the main program we set the total number of runs to be 30 and the
number of simultaneous tasks to maxq=3.

We go into the while loop checking if the number of tasks in future
is less than maxq and we have not launched our whole set of runs.   We
print when we have added work.

We then check our list of tasks in future for ones that are done.  If
it is done we print the result and remove it from the list of work.

Note that the call to done is nonblocking.

The srun commands shown at the bottom is for running with mpi4py with
IntelMPI as the backend MPI inside of a interactive session.
"""
```

queue.py

```
#!/usr/bin/env python
import mpi4py
from mpi4py.futures import MPIPoolExecutor
from random import random,randint



def ranword():
    back=""
    for j in range(0,20) :
        set=random()
        if set < 1./3 :
            myrange=[33,64]
        else:
            if set > 2./3. :
                myrange=[65,90]
            else:
                myrange=[97,122]
        i=randint(myrange[0],myrange[1])
        back=back+chr(i)
    return back


def task(*args, **kargs) :
    import time
    myval = 0
    # Iterating over the Python args tuple
    st=time.asctime()
    astr=""
    for x in args:
        astr=astr+str(x)+" "
    time.sleep(3+5*random())
    et=time.asctime()
    return st+" "+et+" "+astr+" "+mpi4py.MPI.Get_processor_name()
```

queue.py

```
if __name__ == '__main__' :
    import time
    now1=time.time()
    print("start time ",time.asctime())
    # our dictionary to hold things in progress
    future={}
    # number of runs to do
    nruns=30
    # maximum number to be running at one time
    maxq=3
    done = 0
    started = 0
    with MPIPoolExecutor() as executor:
        while done < nruns :
            if len(future) < maxq  and  started < nruns :
                # our input is just a random word, could be a list of inputs or directories
                x=ranword()
                # put the work in the queue
                future[x]=executor.submit(task,x,started)
                started=started+1
                print("added work ",done,started,x)
            # check our list for done work
            for key in  list(future) :
                f=future[key]
                if f.done() :
                    # print result and remove it from the dictionary
                    done = done+1
                    print(f.result(),started,done)
                    del future[key]
    now2=time.time()
    print(" end time ",time.asctime(),now2-now1)
```

queue.py

```
srun  --x11 --account=hpcapps --time=1:00:00 --partition=debug --ntasks=8 --nodes=1 --pty bash
module load conda

source activate dompi
module load mpt
srun -u -n 4 python -m mpi4py.futures ./queue.py
```

queue.py

```
(/home/tkaiser2/.conda-envs/dompi) r1i7n35:mpi4py> srun -u -n 4 python -m mpi4py.futures ./queue.py
start time  Mon Jan 11 09:20:28 2021
added work  0 1 .$/8*G7KUr5NpAY2ma6M
added work  0 2 *iE%aW-ahvLg,C=Xw=dX
added work  0 3 7Seqs&Mwn+'e<B%gI0=c
Mon Jan 11 09:20:28 2021 Mon Jan 11 09:20:35 2021 .$/8*G7KUr5NpAY2ma6M 0  r1i7n35 3 1
added work  1 4 6JF;Ax%+HBC8dY4u#vv!
Mon Jan 11 09:20:28 2021 Mon Jan 11 09:20:35 2021 7Seqs&Mwn+'e<B%gI0=c 2  r1i7n35 4 2
added work  2 5 mMoUFeEA$3kW(4P-BB*N
Mon Jan 11 09:20:28 2021 Mon Jan 11 09:20:35 2021 *iE%aW-ahvLg,C=Xw=dX 1  r1i7n35 5 3
added work  3 6 e6PWr3"z@RK'bh"X+oL$
Mon Jan 11 09:20:35 2021 Mon Jan 11 09:20:38 2021 mMoUFeEA$3kW(4P-BB*N 4  r1i7n35 6 4
added work  4 7 $WQngfJT1?;T3U9YvsP@
Mon Jan 11 09:20:35 2021 Mon Jan 11 09:20:42 2021 e6PWr3"z@RK'bh"X+oL$ 5  r1i7n35 7 5
added work  5 8 Fp7Z?VED#Pvb>@-/JV/f
Mon Jan 11 09:20:35 2021 Mon Jan 11 09:20:42 2021 6JF;Ax%+HBC8dY4u#vv! 3  r1i7n35 8 6
added work  6 9 >74d5adlS3E=P+$hr!r$
Mon Jan 11 09:20:38 2021 Mon Jan 11 09:20:44 2021 $WQngfJT1?;T3U9YvsP@ 6  r1i7n35 9 7
added work  7 10 Nv!Px:"4MC+82H)p:ccQ
Mon Jan 11 09:20:44 2021 Mon Jan 11 09:20:48 2021 Nv!Px:"4MC+82H)p:ccQ 9  r1i7n35 10 8
added work  8 11 M;2Gh>;ZkyaDllKPz"=z
Mon Jan 11 09:20:42 2021 Mon Jan 11 09:20:49 2021 >74d5adlS3E=P+$hr!r$ 8  r1i7n35 11 9
added work  9 12 r7O6L10aHquAYA=of?)w
Mon Jan 11 09:20:42 2021 Mon Jan 11 09:20:50 2021 Fp7Z?VED#Pvb>@-/JV/f 7  r1i7n35 12 10
added work  10 13 QCv#MMp&,pxxe:p7vSQy
Mon Jan 11 09:20:49 2021 Mon Jan 11 09:20:52 2021 M;2Gh>;ZkyaDllKPz"=z 10  r1i7n35 13 11
added work  11 14 RN25neu:qDuOFD0aa$A$
Mon Jan 11 09:20:50 2021 Mon Jan 11 09:20:56 2021 QCv#MMp&,pxxe:p7vSQy 12  r1i7n35 14 12
added work  12 15 xob5TO,THRHCTU6uJ-J)
Mon Jan 11 09:20:49 2021 Mon Jan 11 09:20:57 2021 r7O6L10aHquAYA=of?)w 11  r1i7n35 15 13
added work  13 16 c8:o&1GSCtg=vp&XgH)o
Mon Jan 11 09:20:52 2021 Mon Jan 11 09:20:59 2021 RN25neu:qDuOFD0aa$A$ 13  r1i7n35 16 14
added work  14 17 n&?D/ZM>PMCODeW<IMdr
Mon Jan 11 09:20:56 2021 Mon Jan 11 09:21:01 2021 xob5TO,THRHCTU6uJ-J) 14  r1i7n35 17 15
added work  15 18 #J7Eqh'Ii;I*x76z>mgW
Mon Jan 11 09:20:57 2021 Mon Jan 11 09:21:03 2021 c8:o&1GSCtg=vp&XgH)o 15  r1i7n35 18 16
added work  16 19 &nD%Lxqm.j9avwPA=7b&
Mon Jan 11 09:20:59 2021 Mon Jan 11 09:21:06 2021 n&?D/ZM>PMCODeW<IMdr 16  r1i7n35 19 17
added work  17 20 bOoMY4L-AVjZnUFVo)gJ
Mon Jan 11 09:21:01 2021 Mon Jan 11 09:21:06 2021 #J7Eqh'Ii;I*x76z>mgW 17  r1i7n35 20 18
added work  18 21 -Aq%d3eM>nY-pejMw@aE
Mon Jan 11 09:21:03 2021 Mon Jan 11 09:21:07 2021 &nD%Lxqm.j9avwPA=7b& 18  r1i7n35 21 19
added work  19 22 c'oCJLV:IrT4?oUTf(g?
Mon Jan 11 09:21:07 2021 Mon Jan 11 09:21:11 2021 c'oCJLV:IrT4?oUTf(g? 21  r1i7n35 22 20
added work  20 23 Ng"vLYK$U42Ubx.b9.LZ
Mon Jan 11 09:21:06 2021 Mon Jan 11 09:21:11 2021 -Aq%d3eM>nY-pejMw@aE 20  r1i7n35 23 21
added work  21 24 py<"g>%lX,dUvFz7m:O@
Mon Jan 11 09:21:06 2021 Mon Jan 11 09:21:14 2021 bOoMY4L-AVjZnUFVo)gJ 19  r1i7n35 24 22
added work  22 25 (0S>T4Nr@aq=3)u':JaP
Mon Jan 11 09:21:11 2021 Mon Jan 11 09:21:14 2021 Ng"vLYK$U42Ubx.b9.LZ 22  r1i7n35 25 23
added work  23 26 gjcbIDVUk8Fbs:"hMXXT
Mon Jan 11 09:21:11 2021 Mon Jan 11 09:21:18 2021 py<"g>%lX,dUvFz7m:O@ 23  r1i7n35 26 24
added work  24 27 lN@CMIYt=-TehJrW(.oO
Mon Jan 11 09:21:14 2021 Mon Jan 11 09:21:19 2021 (0S>T4Nr@aq=3)u':JaP 24  r1i7n35 27 25
added work  25 28 e0@HW5%ko'Xn)P"T?KuY
Mon Jan 11 09:21:14 2021 Mon Jan 11 09:21:20 2021 gjcbIDVUk8Fbs:"hMXXT 25  r1i7n35 28 26
added work  26 29 H@0$uSpEHm7"y6/OLD7I
Mon Jan 11 09:21:19 2021 Mon Jan 11 09:21:24 2021 e0@HW5%ko'Xn)P"T?KuY 27  r1i7n35 29 27
added work  27 30 h(stLTWo5OS7d09=20Vz
Mon Jan 11 09:21:18 2021 Mon Jan 11 09:21:26 2021 lN@CMIYt=-TehJrW(.oO 26  r1i7n35 30 28
Mon Jan 11 09:21:20 2021 Mon Jan 11 09:21:28 2021 H@0$uSpEHm7"y6/OLD7I 28  r1i7n35 30 29
Mon Jan 11 09:21:24 2021 Mon Jan 11 09:21:28 2021 h(stLTWo5OS7d09=20Vz 29  r1i7n35 30 30
 end time  Mon Jan 11 09:21:28 2021 59.94140338897705
(/home/tkaiser2/.conda-envs/dompi) r1i7n35:mpi4py>
```

Dask

- Dask is a flexible library for parallel computing in Python.
- https://docs.dask.org/en/latest/
- Kevin Sayers

~/examples/bot/dask/daskit.py

```
#!/usr/bin/env python

import time
import os
import sys
import numpy as np
import socket
from distributed import Client, LocalCluster
import dask
from collections import Counter


# get our start time
global st
st=time.time()
```

```
def test(i,j=10):
# get pid, start time, host  and sleep for j seconds
   pid=os.getpid()
   when=time.time()-st
   print("%6d   %6.2f" % (pid,when))
   time.sleep(j)
   back="%s %6.2f %s" % (str(os.getpid()),when,socket.gethostname())
   return back
```

~/examples/bot/dask/daskit.py

```
def main():
#get command line arguments controling launch
   threads=1
   workers=8
   for x in sys.argv[1:] :
     if x.find("threads") > -1 :
        z=x.split("=")
        threads=int(z[1])
     if x.find("workers") > -1 :
        z=x.split("=")
        workers=int(z[1])

# launch with either threads and/or workers specified (0 = default)
   if threads == 0 and workers != 0 :
      print("lanching  %d workers, default threads" % (workers))
      cluster = LocalCluster(n_workers=workers)
   if threads != 0 and workers == 0 :
      print("lanching  %d threads, defalut workers" % (threads))
      cluster = LocalCluster(threads_per_worker=threads)
   if threads != 0 and workers != 0 :
      print("lanching  %d workers  with %d threads" % (workers,threads))
      cluster = LocalCluster(n_workers=workers,threads_per_worker=threads)
   print(cluster)
   client = Client(cluster)
   print(client)
```

~/examples/bot/dask/daskit.py

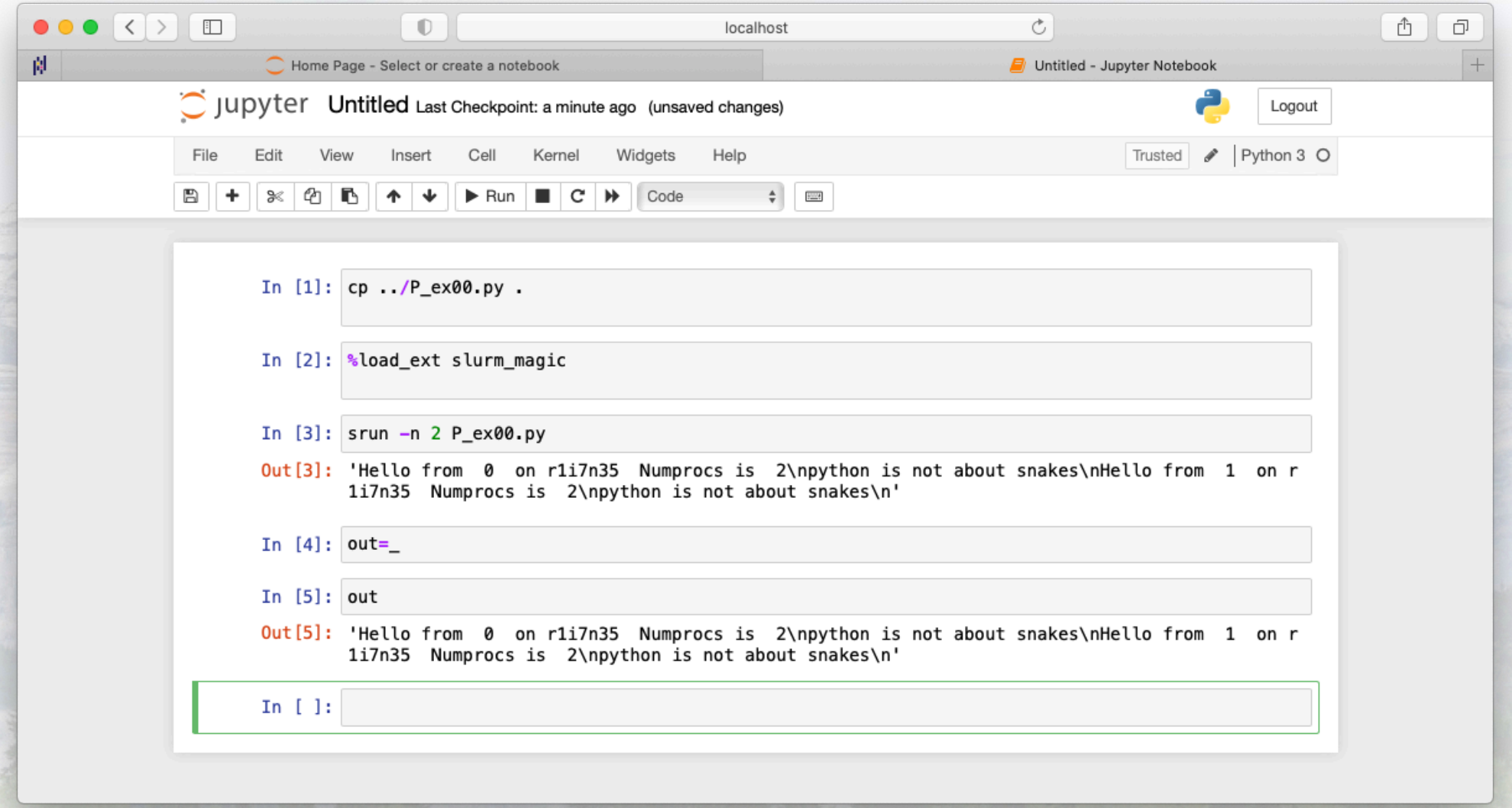
```
# do serial
# NOTE: it is possible to launch an asynchronous client
# but here we just do serial synchronous.  See:
# https://distributed.dask.org/en/latest/asynchronous.html
    result = []
    print("   pid  Start T")
    for i in range (0,5):
        j=2
        result.append(client.submit(test,i,j).result())
    print(result)
    print (Counter(result))
#do parallel
    n=15
    np.random.seed(1234)
    x=np.random.random(n)*20
#set to uniform nonzero to get uniform run times for each task
    x=np.ones(n)*10
    print(x)
    print("   pid  Start T")
    L=client.map(test,range(n),x)
    mylist=client.gather(L)
    pids=[]
    for m in mylist:
      x=m.split()[0]
      pids.append(x)
      print(m)
    pids=sorted(set(pids))
    print(len(pids),pids)

if __name__ == '__main__':
    main()
```

```
./daskit.py threads=2 workers=4
```

Jupyter Slurm_magic commands

- https://github.com/NERSC/slurm-magic
- pip install git+git://github.com/NERSC/slurm-magic.git
- In a notebook
  - %load_ext slurm_magic
- Adds: sacct, sacctmgr, salloc, sattach, sbatch, sbcast, scancel, scontrol, sdiag, sinfo, slurm, smap, sprio, squeue, sreport, srun, sshare, sstat, strigger, sview

Run MPI from Jupyter



Some PLEXOS Scripts

- There's a two script collection that has been floating around
  - Run first script to submit a collection of runs using the second
  - Has become a bit bloated
  - Even successful runs are reported as failed
- Goal:
  - Clean it up a bit
  - Add a few new features
  - Do the same thing but with a single script and array jobs.

submitPLEXOS.sh

- Takes as input
  - *xml file and
  - A text file a list of models to run
- Hardwired slurm script
- Runs the slurm script for each model in the list

```
head r3_CO2_150n35_NoCCS2035_HPC.xml
<MasterDataSet xmlns="http://tempuri.org/MasterDataSet.xsd">
  <t_attribute>
    <attribute_id>1</attribute_id>
    <class_id>2</class_id>
    <enum_id>1</enum_id>
    <name>Latitude</name>
    <unit_id>11</unit_id>
    <default_value>0</default_value>
    <is_enabled>false</is_enabled>
    <is_integer>false</is_integer>
el2:atest>
```

```
cat models.txt
model_2035_m1
model_2035_m2
model_2035_m3
model_2035_m4
model_2035_m5
model_2035_m6
model_2035_m7
model_2035_m8
model_2035_m9
model_2035_m10
model_2035_m11
model_2035_m12
el2:atest>
```

submitPLEXOS.sh

```
#!/bin/bash

# Example input to run the models listed in .txt in the .xml input file
# ./submitPLEXOS.sh r3_CO2_150n35_NoCCS2035_HPC.xml models.txt

############################ USER MODIFIED SECTION ######################
export runtime="08:00:00"
export alloc="hpcapps"    #allcoation to use
export runscript="runPLEXOS.sh"
#########################################################################

name="${1%.*}"
models=$2

echo $name
mkdir -p ${name}

rootdir=$(pwd)
cd $rootdir

### set topdir if you want a directory for
### all jobs submitted at then same time
topdir=`date +"%y%m%d%H%M%S"`

while read line; do
if [ -n "$topdir" ] ; then
 export mydir=${topdir}/${name}/${line}
 echo running in $topdir
else
 export mydir=${name}/${line}
fi

# clean out the directory if it exists
 rm -rf ${mydir}
 mkdir -p ${mydir}

 cp "${runscript}" "$mydir/."

 cp "${name}.xml" "$mydir/."

 ln -fs "${rootdir}/data/" "$mydir/."

 cd "$mydir/"

 echo "++++++++"
 pwd
 ls -lt
 echo "++++++++"

 export SLURM_SUBMIT_DIR=$(pwd)
 submitcommand="sbatch -A ${alloc} -t ${runtime} --export=filename="${name}",model="${line}" ${runscript} --qos=medium"
 echo $submitcommand
 $submitcommand
 cd $rootdir

done < $models
```

My additions

Create a top level directory for all jobs submitted at the same time.

Clean out failed runs

- Sets
  - File structure
  - Account
  - Time
  - Xml filename
  - model
- Calls sbatch with specified run script

runPLEXOS.sh

Usage:

```
#!/bin/bash
# This script runs PLEXOS  on  Eagle.
# It needs a separate script that sets variables
# and runs sbatch.
# OR
# export filename=FILENAME
# export model=MODEL
# export time=TIME
# sbatch -A account -p partition -t time ./runPLEXOS.sh
```

runPLEXOS.sh
First addition - check license server

```
# This function tests if the PLEXOS license server can be seen
# It will check 5 times and error out if it is not seen.  It
# does not check if your license file is valid.
license () {
    for attempt in `seq 5` ; do
      ping -c 2 10.60.3.188  > /dev/null
      if [ $? -eq 1 ] ; then
        date
        echo -e "Can not see license server. \nWill try again in 60 seconds.\n"
      else
        return 0
      fi
      if [ $attempt -lt 5 ]  ; then sleep 60 ; fi
    done
    date
    echo "license lookup failed - exiting"
    date                                >> license.fail
    hostname                            >> license.fail
    ping -c 2 10.60.3.188               >> license.fail
    ping -c 2 eagle-dav.hpc.nrel.gov  >> license.fail
    exit
}

# Call the license check function
license
echo "found license server"
```

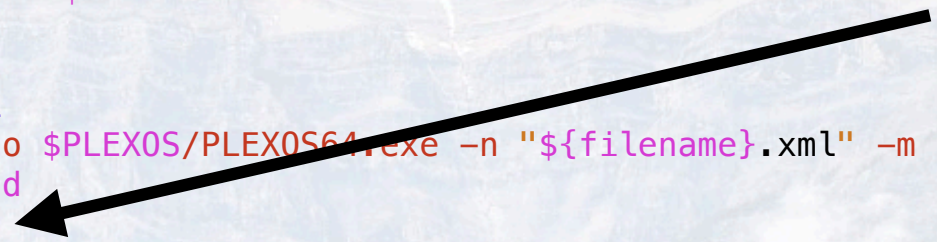
runPLEXOS.sh
Second addition - load version specific modules

```
module use -a /nopt/nrel/apps/modules/default/modulefiles
module purge

# set the version to use - defaults to 8.2 or you
# can set it in the calling environment like you
# do for filename and model
case $version in
    7.4)   # loads PLEXOS 7.4
        module load mono/4.6.2.7 xpressmp/8.0.4 centos plexos/7.400.2
        ;;
    8.1)   # loads PLEXOS 8.1
        module load centos mono xpressmp/8.5.6 plexos/8.100R02
        ;;
    *)     # default to PLEXOS 8.2
        module load centos mono/6.8.0.105 xpressmp/8.5.6 plexos/8.200R01
        version=8.2
        ;;
esac

echo 'I am in ' $PWD ' submitting for PLEXOS ' $version
```

runPLEXOS.sh

Main portion of the script - no major changes here

```
# MAX_TEMP_FILE_AGE is an environmental variable that Plexos uses to determine
# how old a temporary directory can be (in days) before it should be purged
# see http://wiki.energyexemplar.com/index.php?n=Article.AdvancedSettings
export MAX_TEMP_FILE_AGE=50

# PLEXOS_TEMP is an environmental variable that Plexos uses to store temporary files.
# Plexos creates subdirectories in this directory for each run.
# If the subdirectory gets deleted during the run then the run will fail when it
# tries to write the solution file.
export PLEXOS_TEMP=/scratch/$USER/tmp/$SLURM_JOBID
export TEMP=$PLEXOS_TEMP

#make sure the PLEXOS_TEMP and TEMP directories exist
mkdir -p $PLEXOS_TEMP $TEMP


## Run PLEXOS model
plexos_command="mono $PLEXOS/PLEXOS64.exe -n "${filename}.xml" -m "${model}""
echo $plexos_command
$plexos_command

## move zip files (keep in folder by xml so as not to overwrite solutions)
cd ../..
mkdir -p ${filename}_solutions
echo 'Moving zip files'
mv "${filename}/${model}/Model ${model} Solution/Model ${model} Solution.zip"  "${filename}_solutions/
Model ${model} Solution.zip"
```



Line that actually runs things

runPLEXOS.sh
One more minor change to prevent errors on exit

```
## Create h5 files using julia script
## We check if julia with h5plexos is in our
## path and the local file runH5.jl exists
export doh5=0
if [ "$doh5" == 1 ] && [ -a runH5.jl ]  && julia -e "using H5PLEXOS" 2> /dev/null ;then
   echo "Running h5plexos"
   julia runH5.jl ${filename}_solutions ${model}
else
   echo "Skipping h5plexos"
fi
```

Will only run if you have a working julia, the *jl script in your directory and you set the variable doh5

runPLEXOS.sh

```
#!/bin/bash
# This script runs PLEXOS  on  Eagle

# export filename=FILENAME
# export model=MODEL
# export time=TIME
# sbatch -A account -p partition -t time ./runPLEXOS.sh
# Or use a separate script that sets varaibles
# and runs sbatch.

# This function tests if the PLEXOS license server can be seen
# It will check 5 times and error out if it is not seen.  It
# does not check if your license file is valid.
license () {
    for attempt in `seq 5` ; do
      ping -c 2 10.60.3.188  > /dev/null
      if [ $? -eq 1 ] ; then
        date
        echo -e "Can not see license server. \nWill try again in 60 seconds.\n"
      else
        return 0
      fi
      if [ $attempt -lt 5 ]  ; then sleep 60 ; fi
    done
    date
    echo "license lookup failed - exiting"
    date                                >> license.fail
    hostname                            >> license.fail
    ping -c 2 10.60.3.188               >> license.fail
    ping -c 2 eagle-dav.hpc.nrel.gov    >> license.fail
    exit
}

# Call the license check function
license
echo "found license server"
```

runPLEXOS.sh

```
module use -a /nopt/nrel/apps/modules/default/modulefiles
module purge

# set the version to use - defaults to 8.2 or you
# can set it in the calling environment like you
# do for filename and model
case $version in
    7.4)   # loads PLEXOS 7.4
        module load mono/4.6.2.7 xpressmp/8.0.4 centos plexos/7.400.2
        ;;
    8.1)   # loads PLEXOS 8.1
        module load centos mono xpressmp/8.5.6 plexos/8.100R02
        ;;
    *)     # default to PLEXOS 8.2
        module load centos mono/6.8.0.105 xpressmp/8.5.6 plexos/8.200R01
        version=8.2
        ;;
esac

echo 'I am in ' $PWD ' submitting for PLEXOS ' $version

# MAX_TEMP_FILE_AGE is an environmental variable that Plexos uses to determine
# how old a temporary directory can be (in days) before it should be purged
# see http://wiki.energyexemplar.com/index.php?n=Article.AdvancedSettings
export MAX_TEMP_FILE_AGE=50

# PLEXOS_TEMP is an environmental variable that Plexos uses to store temporary files.
# Plexos creates subdirectories in this directory for each run.
# If the subdirectory gets deleted during the run then the run will fail when it
# tries to write the solution file.
export PLEXOS_TEMP=/scratch/$USER/tmp/$SLURM_JOBID
export TEMP=$PLEXOS_TEMP

#make sure the PLEXOS_TEMP and TEMP directories exist
mkdir -p $PLEXOS_TEMP $TEMP
```

runPLEXOS.sh

```
# MAX_TEMP_FILE_AGE is an environmental variable that Plexos uses to determine
# how old a temporary directory can be (in days) before it should be purged
# see http://wiki.energyexemplar.com/index.php?n=Article.AdvancedSettings
export MAX_TEMP_FILE_AGE=50

# PLEXOS_TEMP is an environmental variable that Plexos uses to store temporary files.
# Plexos creates subdirectories in this directory for each run.
# If the subdirectory gets deleted during the run then the run will fail when it
# tries to write the solution file.
export PLEXOS_TEMP=/scratch/$USER/tmp/$SLURM_JOBID
export TEMP=$PLEXOS_TEMP

#make sure the PLEXOS_TEMP and TEMP directories exist
mkdir -p $PLEXOS_TEMP $TEMP


## Run PLEXOS model
plexos_command="mono $PLEXOS/PLEXOS64.exe -n "${filename}.xml" -m "${model}""
echo $plexos_command
$plexos_command

## move zip files (keep in folder by xml so as not to overwrite solutions)
cd ../..
mkdir -p ${filename}_solutions
echo 'Moving zip files'
mv "${filename}/${model}/Model ${model} Solution/Model ${model} Solution.zip"  "${filename}_solutions/Model ${model} Solution.zip"

## Create h5 files using julia script
## We check if julia with h5plexos is in our
## path and the local file runH5.jl exists
export doh5=0
if [ "$doh5" == 1 ] && [ -a runH5.jl ]  && julia -e "using H5PLEXOS" 2> /dev/null ;then
 echo "Running h5plexos"
 julia runH5.jl ${filename}_solutions ${model}
else
  echo "Skipping h5plexos"
fi
```

Array jobs, multiple jobs submitted with a single script.

array.sh

- This script is designed to run array jobs.
- Array jobs are often a collection of similar jobs with different inputs.

- When an slurm runs a collection of array jobs it assigns two additional variables:

```
SLURM_ARRAY_JOB_ID
SLURM_ARRAY_TASK_ID
```

- Syntax for submitting array jobs

```
sbatch --array=1-12 array.sh
```

- SLURM_ARRAY_TASK_ID would be in the range 1-12

- We use the SLURM_ARRAY_TASK_ID to grab one of the 12 lines of our input file.

- This is also legal:

```
sbatch --array=1,4,9,17
```

- We'll grab lines 1,4,9,17

runPLEXOS.sh
Primary additions

```
#!/bin/bash
#SBATCH --out=%J.out
#SBATCH --error=%J.err
#SBATCH --job-name=PLEXOS
#SBATCH --nodes=1
#SBATCH --time=4:00:00
#SBATCH --partition=short

# This script runs PLEXOS  on  Eagle

# export filename=FILENAME
# export LIST=model_list  # defaults to in_list
# sbatch -A account -p partition -t time --array=1-3 ./array.sh
```

```
# get the JOB and SUBJOB ID
if [[ $SLURM_ARRAY_JOB_ID ]] ; then
        export JOB_ID=$SLURM_ARRAY_JOB_ID
        export SUB_ID=$SLURM_ARRAY_TASK_ID
else
        export JOB_ID=$SLURM_JOB_ID
        export SUB_ID=1
fi

# make a top level directory for the job
# if it does not already exist
mkdir -p $JOB_ID
cd $JOB_ID
```

```
if [ -z ${LIST+x} ]; then echo "LIST is unset"; export LIST=in_list ; else echo "LIST is set to '$LIST'"; fi
export model=`head -n $SUB_ID $SLURM_SUBMIT_DIR/$LIST | tail -1`
```

```
# make a directory for the subjob and go there
#mkdir -p $SUB_ID
#cd $SUB_ID
mkdir -p $model
cd $model
# Make a copy of our script
cat $0 > myscript
printenv > variables
```

Submission comparison

```
el2:atest> cat models.txt
model_2035_m1
model_2035_m2
model_2035_m3
el2:atest> ./submitPLEXOS.sh r3_CO2_150n35_NoCCS2035_HPC.xml models.txt
r3_CO2_150n35_NoCCS2035_HPC
running in 210201093000
++++++++
/scratch/tkaiser2/shared/atest/210201093000/r3_CO2_150n35_NoCCS2035_HPC/model_2035_m1
total 1
lrwxrwxrwx. 1 tkaiser2 tkaiser2       36 Feb  1 09:30 data -> /scratch/tkaiser2/shared/atest/data/
-rwx------. 1 tkaiser2 tkaiser2 38433842 Feb  1 09:30 r3_CO2_150n35_NoCCS2035_HPC.xml
-rw-rw----. 1 tkaiser2 tkaiser2     3071 Feb  1 09:30 runPLEXOS.sh
++++++++
sbatch -A hpcapps -t 08:00:00 --export=filename=r3_CO2_150n35_NoCCS2035_HPC,model=model_2035_m1 runPLEXOS.sh --qos=medium
Submitted batch job 5790282
running in 210201093000
++++++++
/scratch/tkaiser2/shared/atest/210201093000/r3_CO2_150n35_NoCCS2035_HPC/model_2035_m2
total 16385
lrwxrwxrwx. 1 tkaiser2 tkaiser2       36 Feb  1 09:30 data -> /scratch/tkaiser2/shared/atest/data/
-rwx------. 1 tkaiser2 tkaiser2 38433842 Feb  1 09:30 r3_CO2_150n35_NoCCS2035_HPC.xml
-rw-rw----. 1 tkaiser2 tkaiser2     3071 Feb  1 09:30 runPLEXOS.sh
++++++++
sbatch -A hpcapps -t 08:00:00 --export=filename=r3_CO2_150n35_NoCCS2035_HPC,model=model_2035_m2 runPLEXOS.sh --qos=medium
Submitted batch job 5790283
running in 210201093000
++++++++
/scratch/tkaiser2/shared/atest/210201093000/r3_CO2_150n35_NoCCS2035_HPC/model_2035_m3
total 16385
lrwxrwxrwx. 1 tkaiser2 tkaiser2       36 Feb  1 09:30 data -> /scratch/tkaiser2/shared/atest/data/
-rwx------. 1 tkaiser2 tkaiser2 38433842 Feb  1 09:30 r3_CO2_150n35_NoCCS2035_HPC.xml
-rw-rw----. 1 tkaiser2 tkaiser2     3071 Feb  1 09:30 runPLEXOS.sh
++++++++
sbatch -A hpcapps -t 08:00:00 --export=filename=r3_CO2_150n35_NoCCS2035_HPC,model=model_2035_m3 runPLEXOS.sh --qos=medium
Submitted batch job 5790284
el2:atest>
el2:atest>
```

Submission comparison

```
el2:atest> cat in_list
cat in_list
model_2035_m1
model_2035_m2
model_2035_m3
model_2035_m4
model_2035_m5
model_2035_m6
model_2035_m7
model_2035_m8
model_2035_m9
model_2035_m10
model_2035_m11
model_2035_m12
el2:atest>

el2:atest> export filename=r3_CO2_150n35_NoCCS2035_HPC
el2:atest> sbatch -A hpcapps -p short -t 4:00:00 --array=10-12 ./array.sh
Submitted batch job 5790650


el2:atest> squeue -u tkaiser2
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
        5790656_10     short   PLEXOS tkaiser2  R       0:44      1 r5i7n12
        5790656_11     short   PLEXOS tkaiser2  R       0:44      1 r8i1n28
        5790656_12     short   PLEXOS tkaiser2  R       0:44      1 r2i1n6
           5790283  standard runPLEXO tkaiser2  R      22:50      1 r7i7n14
           5790284  standard runPLEXO tkaiser2  R      22:50      1 r7i7n26
           5790282  standard runPLEXO tkaiser2  R      23:03      1 r1i7n32
el2:atest>
```

Example runH5.jl

```
# julia script for running H5PLEXOS

result = ARGS[1]
model = ARGS[2]

# change working directory to solution folder
cd(result)

# run h5plexos
using H5PLEXOS
input = string("Model ", model, " Solution.zip")
output = string("Model ", model, " Solution.h5")

process(input, output)
```