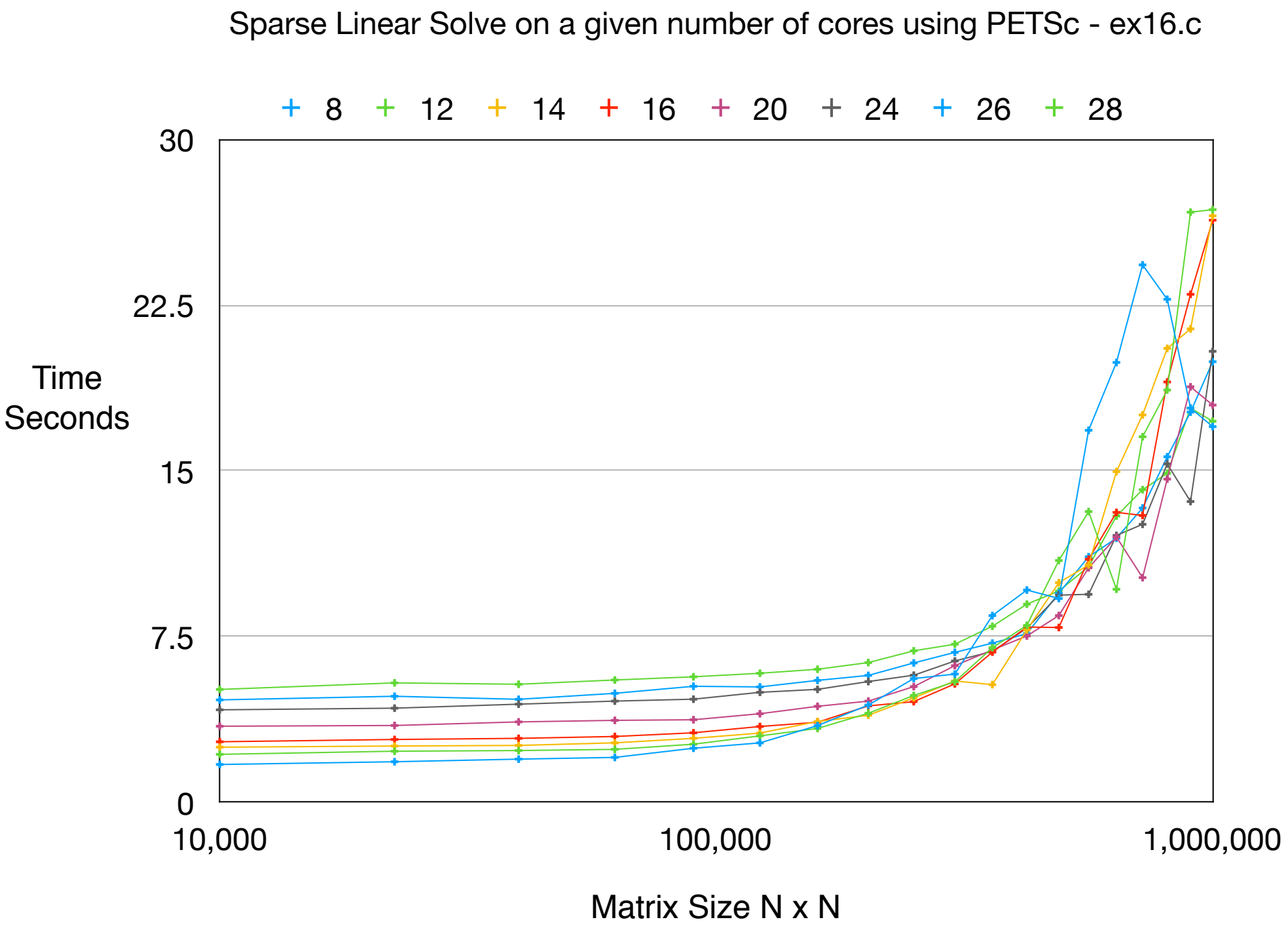
Batch Scripting for Parallel Systems

Author
Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu


Sparse Linear Solve on a given number of cores using PETSc - ex16.c
8
12
14
16
20
24
26
28
30
22.5
15
7.5
0
Time
Seconds
10,000
100,000
1,000,000
Matrix Size N x N

Sparse Linear Solve on a given number of cores using PETSc - ex16.c

Size (M x M)	8	12	14	16	20	24	26	28
10,000	1.7	2.2	2.5	2.7	3.4	4.2	4.6	5.1
22,500	1.8	2.3	2.6	2.8	3.5	4.3	4.8	5.4
40,000	2.0	2.3	2.6	2.9	3.6	4.4	4.7	5.3
62,500	2.0	2.4	2.7	3.0	3.7	4.6	4.9	5.5
90,000	2.5	2.6	2.9	3.2	3.7	4.7	5.3	5.7
122,500	2.7	3.0	3.1	3.4	4.0	5.0	5.2	5.8
160,000	3.5	3.3	3.7	3.6	4.3	5.1	5.5	6.0
202,500	4.4	4.0	3.9	4.4	4.6	5.5	5.7	6.3
250,000	5.6	4.8	4.7	4.6	5.2	5.8	6.3	6.9
302,500	5.8	5.5	5.5	5.4	6.2	6.4	6.8	7.2
360,000	8.5	7.0	5.3	6.8	6.9	6.8	7.2	8.0
422,500	9.6	8.0	7.8	7.9	7.5	7.9	7.6	9.0
490,000	9.2	10.9	9.9	7.9	8.5	9.4	9.5	9.6
562,500	16.8	13.2	10.7	11.0	10.6	9.4	11.1	10.7
640,000	19.9	9.6	15.0	13.1	12.0	12.1	12.0	13.0
722,500	24.3	16.5	17.5	13.0	10.2	12.6	13.3	14.1
810,000	22.8	18.7	20.5	19.0	14.6	15.3	15.6	14.9
902,500	17.8	26.7	21.4	23.0	18.8	13.6	17.7	17.8
1,000,000	17.0	26.8	26.5	26.3	18.0	20.4	19.9	17.2

Purpose:

- To give you some ideas about what is possible
- To give you some examples to follow to help you write your own script
- Be a reference, where we can point people
- Document responses to questions

To Cover...

- Our test codes
- Bash useful concepts
- Basic Scripts
- Using Variables in Scripts
- Redirecting Output, getting output before a job finishes
- Getting Notifications
- Keeping a record of what you did
- Creating directories on the fly for each job
- Using local disk space

To Cover...

- Multiple jobs on a node
  - Sequential
  - Multiple scripts - one node
  - One Script - different MPI jobs on different cores

To Cover...

- Mapping tasks to nodes
  - Less than N tasks per node
  - Different executables working together
  - Hybrid MPI/OpenMP jobs (MPI and Threading)
  - Running on heterogeneous nodes using all cores
- Job dependencies
  - Chaining jobs
  - Jobs submitting new jobs

The Source and Scripts

These programs vary from a glorified "Hello World" to being very complex

We also include copies of all our scripts

http://geco.mines.edu/guide/scripts

```
[joeuser@mio tests]$ [joeuser@mio tests]$ wget http://geco.mines.edu/scripts/morescripts.tgz
[joeuser@mio tests]$ [joeuser@mio tests]$ tar -xzf morescripts.tgz
[joeuser@mio tests]$ [joeuser@mio tests]$ cd morescripts
```

```
[joeuser@mio somescripts]$ make  2> /dev/null
mpicc -o c_ex00 c_ex00.c
mpif90 -o f_ex00 f_ex00.f
rm -rf fmpi.mod
icc info.c -o info_c
ifort info.f90 -o info_f
cp info.py info_p
chmod 700 info_p
ifort -O3 -mkl -openmp pointer.f90 -o fillmem
od -vAn -d -N1048576 < /dev/urandom > segment
tar -czf data.tgz segment
rm -rf segment*
mpicc -DDO_LOCAL_FILE_TEST -c sinkfile.c
mpif90 sinkf.f90 sinkfile.o -o sinkf
mpicc -DDO_LOCAL_FILE_TEST -DDO_C_TEST sinkfile.c -o sinkfile
rm *o *mod
chmod 700 nodes
```

What we have

- [c_ex00.c, c_ex00]
  - hello world in C and MPI
- [f_ex00.f, f_ex00]
  - hello world in Fortran and MPI
- [info.c, info_c] [info.f90, info_f] [info.py]
  - Serial programs in C, Fortran and Python that print the node name and process id. Creates a node name process id

info.c

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
main() {
    char name[128],fname[128];
    pid_t mypid;
    FILE *f;
    char aline[128];

/* get the process id */
    mypid=getpid();
/* get the host name */
    gethostname(name, 128);
/* make a file name based on these two */
    sprintf(fname,"%s_%8.8d",name,(int)mypid);
/* open and write to the file */
    f=fopen(fname,"w");
    fprintf(f,"C says hello from %d on %s\n",(int)mypid,name);
}
```

info.f90

```
program info
  USE IFPOSIX ! needed by PXFGETPID
  implicit none
  integer ierr,mypid
  character(len=128) :: name,fname
! get the process id
  CALL PXFGETPID (mypid, ierr)
! get the node name
  call mynode(name)
! make a filename based on the two
  write(fname,'(a,"_",i8.8)')trim(name),mypid
! open and write to the file
  open(12,file=fname)
  write(12,*)"Fortran says hello from",mypid," on ",trim(name)
end program
```

```
subroutine mynode(name)
! Intel Fortran subroutine to return
! the name of a node on which you are
! running
  USE IFPOSIX
  implicit none
  integer jhandle
  integer ierr,len
  character(len=128) :: name
  CALL PXFSTRUCTCREATE ("utsname", jhandle, ierr)
  CALL PXFUNAME (jhandle, ierr)
  call PXFSTRGET(jhandle,"nodename",name,len,ierr)
end subroutine
```

info.py

```
#!/usr/bin/env python
import os
# get the process id
mypid=os.getpid()
# get the node name
name=os.uname()[1]
# make a filename based on the two
fname="%s_%8.8d" % (name,mypid)
# open and write to the file
f=open(fname,"w")
f.write("Python says hello from %d on %s\n" %(mypid,name))
```

Example Output From the Serial Programs

```
[joeuser@mio cwp]$ ./info_c
[joeuser@mio cwp]$ ls -lt mio*
-rw-rw-r-- 1 joeuser joeuser 41 Jan 11 13:47
mio.mines.edu_00050205
[joeuser@mio cwp]$ cat mio.mines.edu_00050205
C says hello from 50205 on mio.mines.edu
[joeuser@mio cwp]$
```

C MPI example

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>

/*************************************************************
This is a simple hello world program. Each processor prints out
it's rank and the size of the current MPI run (Total number of
processors).
*************************************************************/
int main(argc,argv)
int argc;
char *argv[];
{
    int myid, numprocs,mylen;
    char myname[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(myname,&mylen);

/* print out my rank and this run's PE size*/
    printf("Hello from %d of %d on %s\n",myid,numprocs,myname);

    MPI_Finalize();
}
```

Fortran MPI example

```
 !****************************************
!  This is a simple hello world program. Each processor
!  prints out its rank and total number of processors
!  in the current MPI run.
!*******************************************************
      program hello
      include "mpif.h"
      character (len=MPI_MAX_PROCESSOR_NAME):: myname
      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
      call MPI_Get_processor_name(myname,mylen,ierr)
      write(*,*)"Hello from ",myid," of ",numprocs," on ",trim(myname)
      call MPI_FINALIZE(ierr)
      stop
      end
```

Fortran Matrix Inversion Example

- Fills a number of arrays with random data
- Does a matrix inversion
- Used to test the performance of individual cores of a processor
- Can also be used to test threading

```
      include 'mkl_vsl.fi'
...
      program testinvert
      use numz
...
      call my_clock(cnt1(i))
      CALL DGESV( N, NRHS, twod, LDA, IPIVs(:,i), Bs(:,i), LDB, INFOs(i) )
      call my_clock(cnt2(i))
      write(*,'(i5,i5,3(f12.3))')i,infos(i),cnt2(i),cnt1(i),real(cnt2(i)-cnt1(i),b8)
...
```

sinkfile.c

- A somewhat complicated example
- Does a parallel file copy
  - Copies a file seen by MPI task 0 to
  - Each node (not task) in an MPI program that does not share the node used by task 0
  - Used in a situation where MPI tasks might not share file systems

Batch Scripts

- Batch scripts are just that - scripts
  - Run with some “shell”, bash, csh, tsh, python
  - Most of the things you can do in a normal script can be done in a batch script
- Some lines in the script are comments to the shell
- Comments can have meaning to the parallel environment
- The parallel environment can define/use variables

Bash

- Default shell on CSM machines
- Used to interact with the machine, run commands
- Bash commands can be run interactively or put in a script file
- A script file is really a "simple"
  - Program
  - List of commands
- We will use bash in our examples but other shells and scripting languages have similar capabilities
- First we discuss some features of bash

http://www.tldp.org/LDP/Bash-Beginners-Guide/html/

Notes on Commands

- > is used to sent output to a file (date > mylisting)
- >> append output to a file (ls >> mylisting)
- >& send output and error output to a file
- The ; can be used to combine multiline commands on a single line. Thus the following are equivalent

```
date ; echo "line 2" ; uptime
```

```
date
echo "line 2"
date
```

Notes on Commands

- Putting commands in ` ` returns the output of a command into a variable
- Can be use create a list with other commands such as “for loops”

```
myf90=`ls *f90`
echo $myf90
doint.f90 fourd.f90 tintel.f90 tp.f90 vect.f90
```

```
np=`expr 3 + 4`
np=`expr $SLURM_NNODES \* 4`
np=`expr $SLURM_NNODES / 4`
```

The command expr with “`” can be used to do integer math

For loops

```
myf90=`ls *f90`
for f in $myf90 ; do file $f ; done
```

doint.f90: ASCII program text
fourd.f90: ASCII program text
tintel.f90: ASCII program text
tp.f90: ASCII program text
vect.f90: ASCII program text

```
myf90=`ls *f90`
for f in $myf90
  do file $f
done
```

```
for (( c=1; c<=5; c++ )); do echo "Welcome $c times..."; done
```

Welcome 1 times...
Welcome 2 times...
Welcome 3 times...
Welcome 4 times...
Welcome 5 times...

```
for c in 1 2 3 4 5; do echo "Welcome $c times..."; done
```

Welcome 1 times...
Welcome 2 times...
Welcome 3 times...
Welcome 4 times...
Welcome 5 times...

```
for c in `seq 1 2 6`; do  echo "Welcome $c times..."; date; done
```

Welcome 1 times...
Tue Jul 31 12:17:11 MDT 2012
Welcome 3 times...
Tue Jul 31 12:17:11 MDT 2012
Welcome 5 times...
Tue Jul 31 12:17:11 MDT 2012

```
for c in `seq 1 2 6`
  do
  echo "Welcome $c times..."
  date
done
```

Combing Operations

Operation	Effect
[! EXPR]	True if **EXPR** is false.
[(EXPR)]	Returns the value of **EXPR**. This may be used to override the normal precedence of operators.
[EXPR1 -a EXPR2]	True if both **EXPR1** and **EXPR2** are true.
[EXPR1 -o EXPR2]	True if either **EXPR1** or **EXPR2** is true.

Test Variable Being Set and "if"

We do this loop 3 times.
(1)"var" not set
(2)"var" set but empty
(3)var set and not empty

```
i= 1
var is unset or empty line1
var is unset or empty line2

i= 2
var is unset or empty line1
var is unset or empty line2
var is set but empty line3

i= 3
var is set and not empty
line4
```

```
for i in 1 2 3 ; do
    echo "i=" $i
    if [ $i == 1 ] ; then unset var ; fi
    if [ $i == 2 ] ; then var="" ; fi
    if [ $i == 3 ] ; then var="abcd" ; fi

    if [ -z "$var" ]  ;     then echo "var is unset or empty line1"; fi
    if [ ! -n "$var" ] ;    then echo "var is unset or empty line2"; fi
    if [ -z "${var-x}" ]  ; then echo "var is set but empty line3"; fi
    if [ -n "$var" ] ;      then echo "var is set and not empty line4"; fi
   echo
done
```

String Tests

```
if test "abc" = "def" ;then echo "abc = def" ; else echo "nope 1" ; fi

if test "abc" != "def" ;then echo "abc != def" ; else echo "nope 2" ; fi

if  [ "abc" \< "def" ];then echo "abc < def" ; else echo "nope 3" ; fi

if  [ "abc" \> "def" ]; then echo "abc > def" ; else echo "nope 4" ; fi

if  [ "abc" \> "abc" ]; then echo "abc > abc" ; else echo "nope 5" ; fi
```

```
nope 1
abc != def
abc < def
nope 4
nope 5
```

String Tests

```
if test "abc" = "def" ;then echo "abc = def" ; else echo "nope 1" ; fi
```

```
nope 1
```

```
if test "abc" != "def" ;then echo "abc != def" ; else echo "nope 2" ; fi
```

```
abc != def
```

```
if  [ "abc" \< "def" ];then echo "abc < def" ; else echo "nope 3" ; fi
```

```
abc < def
```

```
if  [ "abc" \> "def" ]; then echo "abc > def" ; else echo "nope 4" ; fi
```

```
nope 4
```

```
if  [ "abc" \> "abc" ]; then echo "abc > abc" ; else echo "nope 5" ; fi
```

```
nope 5
```

File Tests

Test	Meaning
`[ -a FILE ]`	True if `FILE` exists.
`[ -b FILE ]`	True if `FILE` exists and is a block-special file.
`[ -c FILE ]`	True if `FILE` exists and is a character-special file.
`[ -d FILE ]`	True if `FILE` exists and is a directory.
`[ -e FILE ]`	True if `FILE` exists.
`[ -f FILE ]`	True if `FILE` exists and is a regular file.
`[ -g FILE ]`	True if `FILE` exists and its SGID bit is set.
`[ -h FILE ]`	True if `FILE` exists and is a symbolic link.
`[ -k FILE ]`	True if `FILE` exists and its sticky bit is set.
`[ -p FILE ]`	True if `FILE` exists and is a named pipe (FIFO).
`[ -r FILE ]`	True if `FILE` exists and is readable.
`[ -s FILE ]`	True if `FILE` exists and has a size greater than zero.
`[ -t FD ]`	True if file descriptor `FD` is open and refers to a terminal.
`[ -u FILE ]`	True if `FILE` exists and its SUID (set user ID) bit is set.
`[ -w FILE ]`	True if `FILE` exists and is writable.
`[ -x FILE ]`	True if `FILE` exists and is executable.
`[ -O FILE ]`	True if `FILE` exists and is owned by the effective user ID.
`[ -G FILE ]`	True if `FILE` exists and is owned by the effective group ID.
`[ -L FILE ]`	True if `FILE` exists and is a symbolic link.
`[ -N FILE ]`	True if `FILE` exists and has been modified since it was last read.
`[ -S FILE ]`	True if `FILE` exists and is a socket.
`[ FILE1 -nt FILE2 ]`	True if `FILE1` has been changed more recently than `FILE2`, or if `FILE1` exists and `FILE2` does not.
`[ FILE1 -ot FILE2 ]`	True if `FILE1` is older than `FILE2`, or is `FILE2` exists and `FILE1` does not.
`[ FILE1 -ef FILE2 ]`	True if `FILE1` and `FILE2` refer to the same device and inode numbers.

Checking Terminal Input

```
echo "Do you want to proceed?"
echo -n "Y/N: "
read yn
if [ $yn = "y" ] || [ $yn = "Y" ] ; then

  echo "You said yes"

else
  echo "You said no"
fi
```

Note spacing in the if statement. It is important!

Testing Return Code & /dev/null

- Commands return an exit code
  - 0 = success
  - not 0 = failure
- The exit code from the previous command is stored in $?
- $? can be echoed or tested
- This is often used with piping output into /dev/null "the bit bucket" when you only want to know if a command was successful

```
ls a_dummy_file >& /dev/null

if [ $? -eq 0 ] ; then
    echo "ls of a_dummy_file successful"
fi
```

While and with a Test and break

```
rm -f a_dummy_file
while true ; do
  ls a_dummy_file >& /dev/null
  if [ $? -eq 0 ] ; then
    echo "ls of a_dummy_file successful"
  else
    echo "ls of a_dummy_file failed"
  fi
  if [ -a a_dummy_file ] ; then
    echo "a_dummy_file exists, breaking"
    break
  else
    echo "a_dummy_file does not exist"
  fi
  touch a_dummy_file
  echo ; echo "bottom of while loop" ; echo
done
```

```
ls of a_dummy_file failed
a_dummy_file does not exist

bottom of while loop

ls of a_dummy_file successful
a_dummy_file exists, breaking
```

Running Batch Scripts

- A batch script is submitted to a scheduler
  - pbs/torque/moab, sge, lsf, poe,**slurm**
  - Commands to submit scripts
    - qsub, msub, bsub, poe,**sbatch**
- The scheduler decides where and when to run your script
  - Wait for nodes to become available
  - Wait for other jobs to finish
  - Jobs are given a name so that you can track them in the system

Related Commands SLURM

Command	Description - From http://slurm.schedmd.com/man_index.html
sbatch	Submit a batch script to SLURM.
srun	Run parallel jobs
scancel	Used to signal (cancel) jobs or job steps that are under the control of Slurm.
salloc	Obtain a SLURM job allocation (a set of nodes), Useful for interactive sessions.
sacct	Displays accounting data for all jobs and job steps in the SLURM job accounting log or SLURM database
sacctmgr	Used to view and modify Slurm account information.
sattach	Attach to a SLURM job step.
sdiag	scheduling diagnostic tool.
sinfo	view information about SLURM nodes and partitions.
smap	graphically view information about SLURM jobs, partitions, and set configurations parameters.
sprio	view the factors that comprise a job's scheduling priority
squeue	view information about jobs located in the SLURM scheduling queue.
sreport	Generate reports from the slurm accounting data.
sstat	Display various status information of a running job/step.

CSM Unique SLURM Commands

Command /opt/utility/ *	**Description**
sjobs	Summary of running and queued jobs
slurmjobs	Show full information for all jobs -h for help
slurmnodes	Show full information for all nodes (-h for help
inuse	Node usage by group
match	Creates an mpiexec “appfile” for MPMD runs and nonstandard mappings of tasks to nodes
match_split	Creates an srun “multi-prog” for MPMD runs and nonstandard mappings of tasks to nodes
phostname	Glorified MPI/OpenMP “hello world”
mapping	Shows nodes, tasks, threads, cores currently in use by you

phostname “help”

```
phostname arguments:
          -h : Print this help message

no arguments : Print a list of the nodes on which the command is run.

 -f or -1    : Same as no argument but print MPI task id and Thread id
               If run with OpenMP threading enabled OMP_NUM_THREADS > 1
               there will be a line per MPI task and Thread.

 -F or -2    : Add columns to tell first MPI task on a node and and the
               numbering of tasks on a node. (Hint: pipe this output in
               to sort -r

 -a          : Print a listing of the environmental variables passed to
               MPI task. (Hint: use the -l option with SLURM to prepend MPI
               task #.)

 -s ######## : Where ######## is an integer.  Sum a bunch on integers to slow
               down the program.  Should run faster with multiple threads.
```

export OMP_NUM_THREADS=4
srun --nodes 2 -n 4 /opt/utility/phostname.openmpi_3.0.0 -F

```
task    thread             node name  first task    # on node  core
0000      0000           compute176          0000          0000  0000
0000      0001           compute176          0000          0000  0000
0000      0002           compute176          0000          0000  0000
0000      0003           compute176          0000          0000  0000
0001      0000           compute176          0000          0001  0012
0001      0001           compute176          0000          0001  0012
0001      0002           compute176          0000          0001  0012
0001      0003           compute176          0000          0001  0012
0002      0000           compute176          0000          0002  0001
0002      0001           compute176          0000          0002  0001
0002      0002           compute176          0000          0002  0001
0002      0003           compute176          0000          0002  0001
0003      0000           compute177          0003          0000  0000
0003      0001           compute177          0003          0000  0000
0003      0002           compute177          0003          0000  0000
0003      0003           compute177          0003          0000  0000
```

This is not what I expected. How do I fix it?

A Simple Slurm Script for a MPI job

```
#!/bin/bash

#SBATCH --job-name="atest"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH -o stdout
#SBATCH -e stderr
#SBATCH --export=ALL
#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu

#----------------------
cd ~/bins/example/mpi
srun -n 8 ./c_ex00
```

Scripts contain comments designated with a # that are interpreted by SLURM and normal shell commands

We go to this directory

Run this MPI program on 8 cores

A Simple Slurm Script for a MPI job

`#!/bin/bash`	This is a bash script
`#SBATCH --job-name="atest"`	Give our job a name in the
`#SBATCH --nodes=1`	We want I node
`#SBATCH --ntasks-per-node=8`	We expect to run 8 tasks/node
`#SBATCH --time=00:02:00`	We want the node for 2 minutes
`#SBATCH --output=stdout`	Output will go to a file “stdout”
`#SBATCH --error=stderr`	Errors will go to a file “stdout”
`#SBATCH --export=ALL`	**Pass current environment to nodes**
`#SBATCH --mail-type=ALL`	Send email on abort,begin,end
`#SBATCH --mail-user=joeuser@mines.edu`	Address for email
`#---------------------`	Just a normal “comment”
`cd /home/joeuser/examples`	Go to this directory first
`srun -n 8 ./c_ex00`	Run c_ex00 on 8 cores

What happens when you run a script?

- You are given a collection of nodes
- You are logged on to one of the nodes, the primary compute node
- **Any "normal" script command only run on the primary compute node**
- Extra effort must be taken to run on all nodes
- srun or mpiexec (srun for Slurm, mpiexec is used for PBS)
  - Also Run only on the primary compute node
  - Makes the effort to launch MPI jobs on all nodes

Variables in Scripts

Slurm "script" Variables

Variable	Meaning	Typical Value
`SLURM_SUBMIT_DIR`	Directory for the script	/panfs/storage/scratch/joeuser
`SLURM_JOB_USER`	Who are you	joeuser
`SLURM_EXPORT_ENV`	Variables to export	ALL
`SLURM_NNODES`	# nodes for the job	2
`SLURM_JOBID`	Job ID	11160
`SLURM_NODELIST`	Compressed list of nodes	node[114-115]
`SLURM_SUBMIT_HOST`	Host used to launch job	aun002.mines.edu

You can also use variables you define before you submit your script and variables defined in your environment

Example list of variables

```
SLURM_CHECKPOINT_IMAGE_DIR=/bins/joeuser/examples/mpi
SLURM_NODELIST=node[114-115]
SLURM_JOB_NAME=atest
SLURMD_NODENAME=node114
SLURM_TOPOLOGY_ADDR=node114
SLURM_NTASKS_PER_NODE=8
SLURM_PRIO_PROCESS=0
SLURM_NODE_ALIASES=(null)
SLURM_EXPORT_ENV=ALL
SLURM_TOPOLOGY_ADDR_PATTERN=node
SLURM_NNODES=2
SLURM_JOBID=11160
SLURM_NTASKS=16
SLURM_TASKS_PER_NODE=8(x2)
SLURM_JOB_ID=11160
SLURM_JOB_USER=joeuser
SLURM_JOB_UID=15049
SLURM_NODEID=0
SLURM_SUBMIT_DIR=/bins/joeuser/examples/mpi
SLURM_TASK_PID=14098
SLURM_NPROCS=16
SLURM_CPUS_ON_NODE=8
SLURM_PROCID=0
SLURM_JOB_NODELIST=node[114-115]
SLURM_LOCALID=0
SLURM_JOB_CPUS_PER_NODE=8(x2)
SLURM_GTIDS=0
SLURM_SUBMIT_HOST=aun002.mines.edu
SLURM_JOB_PARTITION=aun
SLURM_JOB_NUM_NODES=2
```

A Simple Slurm Script for a MPI job

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH -o stdout
#SBATCH -e stderr
#SBATCH --export=ALL
#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu

#----------------------
cd $SLURM_SUBMIT_DIR
srun -n 8 ./c_ex00
```

We go to "starting" directory

Run this MPI program on 8 cores

docd

A Simple Slurm Script for a MPI job

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j
#SBATCH --export=ALL
#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu

#-----------------------
cd $SLURM_SUBMIT_DIR
srun -n 8 ./c_ex00
```

We go to "starting" directory

Run this MPI program on 8 cores

docd

A Simple Slurm Script for a MPI job

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j
#SBATCH --export=ALL
#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu

#-----------------------
cd $SLURM_SUBMIT_DIR
srun -n 8 ./c_ex00 >& myout.$SLURM_JOB_ID
```

The output from the script

Gives the program output as it runs with each run having a unique output file

redirect

A Simple Slurm Script for a MPI job

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j
#SBATCH --export=ALL
#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu
#------------------------
JOBID=`echo $SLURM_JOB_ID`
cd $SLURM_SUBMIT_DIR

srun -n 8 ./c_ex00 > my_out.$JOBID
```

The output from the script

myout.#####

number

Keeping Records
&
Notifications

We want...

- To keep records of what scripts we have run
- To be notified when a script runs
  - Under PBS
    - #PBS -M with -abe
    - Will send a notice at start and stop
  - Under Slurm
    - #SBATCH —mail-type=ALL
    - Produces information emails
  - We want more than the job number
  - We want everything

Records Notifications

How can I record what I did and where?

How can I know when a particular script starts and exactly what is running?

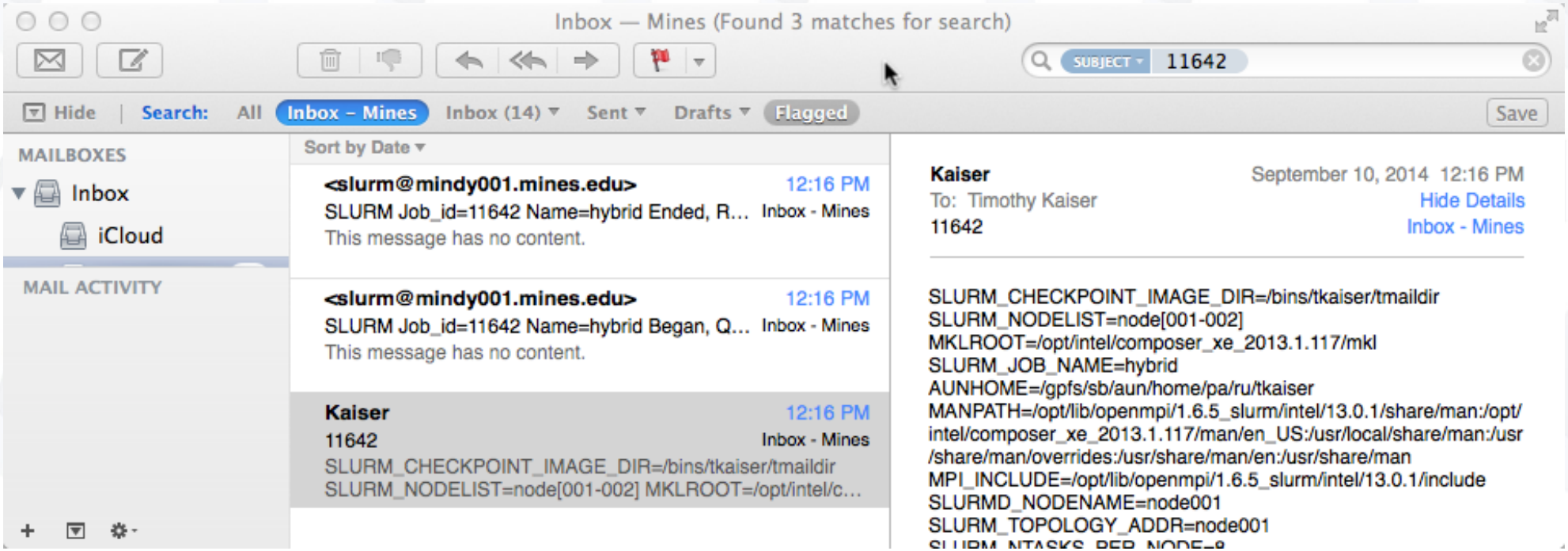
```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu

# 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`

# Save a copy of our environment and script
echo $SLURM_JOB_NODELIST > nodes.$JOBID
cat $0 > script.$JOBID
printenv  > env.$JOBID

#mail us the environment and other "stuff"
####  mail <  env.$JOBID -s $JOBID  $USER@mines.edu
#ssh $SLURM_SUBMIT_HOST "mail <  $MYBASE/$JOBID/env.$JOBID -s $JOBID  $SLURM_JOB_USER@mines.edu"
mkdir -p ~/tmail
cp env.$JOBID ~/tmail
export MAIL_HOST=$SLURM_SUBMIT_HOST
export MAIL_HOST=mindy.mines.edu
ssh mindy.mines.edu "mail <  ~/tmail/env.$JOBID -s $JOBID  $SLURM_JOB_USER@mines.edu"


srun /opt/utility/phostname -F > output.$JOBID
```

Or mio.mines.edu

notify

Lots of records...

```
[joeuser@aun002 tmaildir]$ ls -l *11642*
-rw-rw-r-- 1 joeuser joeuser  6108 Sep 10 12:16 env.11642
-rw-rw-r-- 1 joeuser joeuser    14 Sep 10 12:16 nodes.11642
-rw-rw-r-- 1 joeuser joeuser 15934 Sep 10 12:16 output.11642
-rw-rw-r-- 1 joeuser joeuser  1014 Sep 10 12:16 script.11642
-rw-rw-r-- 1 joeuser joeuser   447 Sep 10 12:16 stderr.11642
-rw-rw-r-- 1 joeuser joeuser     0 Sep 10 12:16 stdout.11642
[joeuser@aun002 tmaildir]$
```



More on variables and a few other details

A digression: /opt/utility/expands

The CSM written utility /opt/utility/expands takes a Slurm style compressed node list and creates a full list similar to what is produces by PBS

```
/opt/utility/expands node[001-003,005-007,100] | sort -u
node001
node002
node003
node005
node006
node007
node100
```

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID

export INPUT=sinput
export APP=fillmemc


/opt/utility/expands $SLURM_JOB_NODELIST >  $APP.$INPUT.nodes.$JOBID
cat $INPUT >  $APP.$INPUT.input.$JOBID
srun ./$APP  < $INPUT >> $APP.$INPUT.output.$JOBID
```

Not an MPI job but we still use srun to ensure that the computation runs on a compute node

env1

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID

export INPUT=sinput
export APP=fillmemc

/opt/utility/expands $SLURM_JOB_NODELIST >  $APP.$INPUT.nodes.$JOBID
cat $INPUT >  $APP.$INPUT.input.$JOBID
srun ./$APP  < $INPUT >> $APP.$INPUT.output.$JOBID
```

Create a short list of the nodes used in my job and give it a unique name. We recommend people always do this.

The command "expands" takes a short list of nodes and expands it to a long list. Now we put our list in a file that has application name, file name, and input file as part of the file name

We are going to use variables for both our input file and application names

Save a copy of our input and put the output in its own file

Not an MPI job but we still use srun to ensure that the computation runs on a compute

env1

What we get

```
[joeuser@aun001 memory]$ ls -l *11714*
-rw-rw-r-- 1 joeuser joeuser   11 Sep 10 13:31 fillmemc.sinput.input.11714
-rw-rw-r-- 1 joeuser joeuser  128 Sep 10 13:31 fillmemc.sinput.nodes.11714
-rw-rw-r-- 1 joeuser joeuser 3259 Sep 10 13:31 fillmemc.sinput.output.11714
-rw-rw-r-- 1 joeuser joeuser    8 Sep 10 13:31 nodes.11714
-rw-rw-r-- 1 joeuser joeuser  205 Sep 10 13:31 stderr.11714
-rw-rw-r-- 1 joeuser joeuser    0 Sep 10 13:31 stdout.11714
[joeuser@aun001 memory]$


[joeuser@aun001 memory]$ cat fillmemc.sinput.nodes.11714 | sort -u
node001
[joeuser@aun001 memory]$ cat fillmemc.sinput.nodes.11714 | wc
     16      16     128
[joeuser@aun001 memory]$


[joeuser@aun001 memory]$ cat fillmemc.sinput.input.11714
4096 64 1

[joeuser@aun001 memory]$ head fillmemc.sinput.output.11714
 matrix size=        4096
 copies=          64
 bytes=     8589934592 gbytes=     8.000
 using mkl for inverts
 generating data for run           1  of           1
 generating time=       2.114 threads= 16
 starting inverts
   33    0   48675.423   48671.805       3.618
    9    0   48675.423   48671.805       3.618
   57    0   48675.423   48671.805       3.618
```

Multiple Executables Tricks

- Case 1: Multiple jobs running on the same node at the same time
  - Independent
  - Launched from different scripts
- Case 2: Multiple executables running on the same node at the same time
  - Independent
  - Launched from a single script
- Either case could be serial or MPI
- Case 3: Using mpiexec or srun to launch several serial programs

Case 1: Multiple Scripts

Or the same script several times

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID
/opt/utility/expands $SLURM_JOB_NODELIST > $APP.$INPUT.nodes.$JOBID

export INPUT=sinput
export APP=fillmemc


cat $INPUT >  $APP.$INPUT.input.$JOBID
srun ./$APP  < $INPUT >> $APP.$INPUT.output.$JOBID
```

We have our application name and input file set as a variable

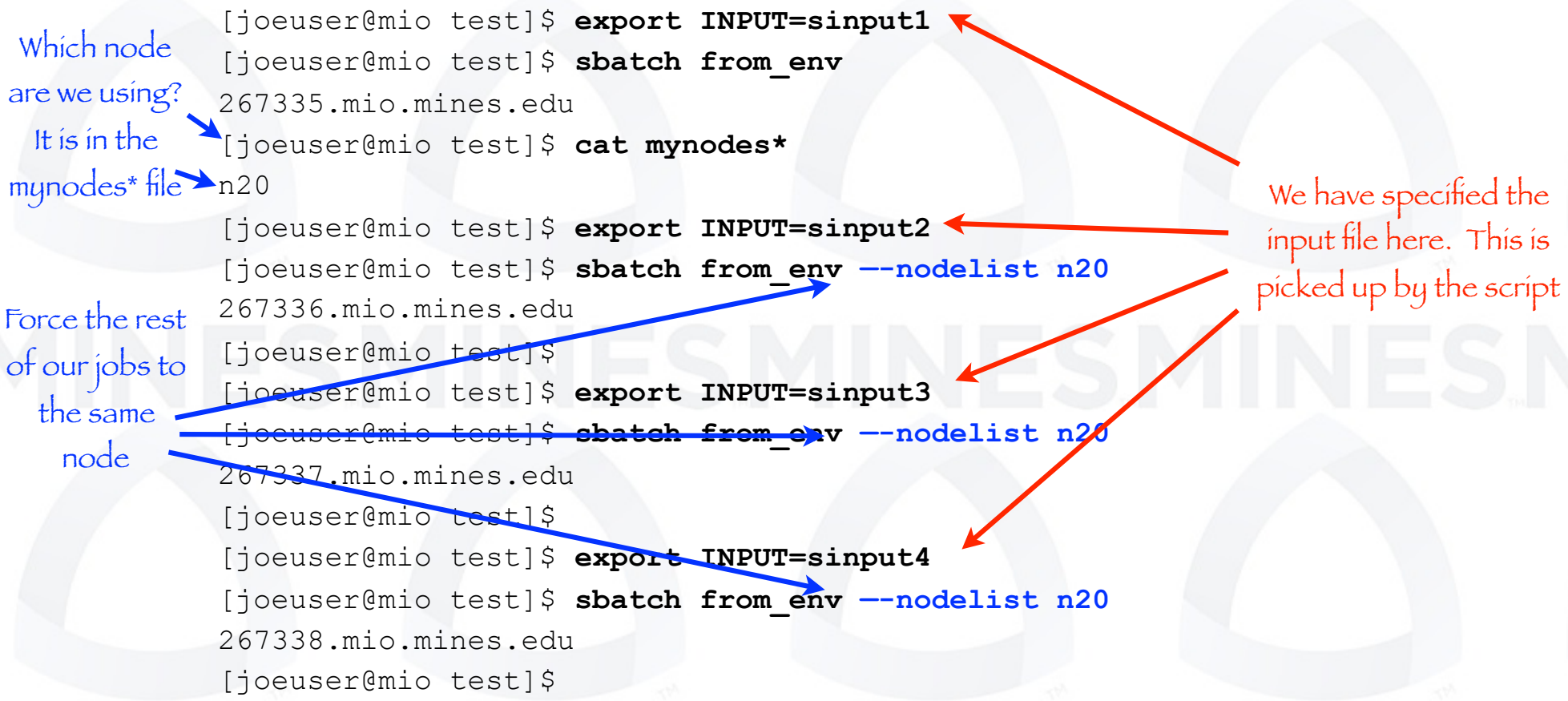
Save a list of nodes, first in just a nodes.* file and then a file that contains the input file name

env1

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --share
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID

#export INPUT=sinput
export APP=fillmemc


/opt/utility/expands $SLURM_JOB_NODELIST >  $APP.$INPUT.nodes.$JOBID
cat $INPUT >  $APP.$INPUT.input.$JOBID
export OMP_NUM_THREADS=2
srun  ./$APP  < $INPUT >> $APP.$INPUT.output.$JOBID
```


We have commented out the line that sets the input file name. We can (must) specify the input before running the script.

Assume we have 4 data sets and we are willing to run on any one node..




```
[joeuser@mio test]$ export INPUT=sinput1
[joeuser@mio test]$ sbatch from_env
267335.mio.mines.edu
[joeuser@mio test]$ cat mynodes*
n20
[joeuser@mio test]$ export INPUT=sinput2
[joeuser@mio test]$ sbatch from_env --nodelist n20
267336.mio.mines.edu
[joeuser@mio test]$
[joeuser@mio test]$ export INPUT=sinput3
[joeuser@mio test]$ sbatch from_env --nodelist n20
267337.mio.mines.edu
[joeuser@mio test]$
[joeuser@mio test]$ export INPUT=sinput4
[joeuser@mio test]$ sbatch from_env --nodelist n20
267338.mio.mines.edu
[joeuser@mio test]$
```

(If you have a reserved node you can specify it for the first run also.)

Our output files:

```
[joeuser@mio test]$ ls -l fillmemc.sinput*

-rw-rw-r-- 1 joeuser joeuser 3395 Feb 15 11:31 fillmemc.sinput1.267335.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 5035 Feb 15 11:32 fillmemc.sinput2.267336.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 6675 Feb 15 11:32 fillmemc.sinput3.267337.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 1541 Feb 15 11:29 fillmemc.sinput4.267334.mio.mines.edu
-rw-rw-r-- 1 joeuser joeuser 1755 Feb 15 11:31 fillmemc.sinput4.267338.mio.mines.edu
[joeuser@mio test]$
```

Note different job numbers

The input file name becomes part of the output file name

Our execution line

```
./$APP  < $INPUT >> $APP.$INPUT.$PBS_JOBID
```

Assume we have 4 data sets and we are willing to run on any one node..

```
[joeuser@aun001 memory]$ export INPUT=sinput1
[joeuser@aun001 memory]$ sbatch  env2
Submitted batch job 11834
[joeuser@aun001 memory]$ cat nodes.11834
node001

[joeuser@aun001 memory]$ export INPUT=sinput2
[joeuser@aun001 memory]$ sbatch --nodelist=node001 env2
Submitted batch job 11835

[joeuser@aun001 memory]$ export INPUT=sinput3
[joeuser@aun001 memory]$ sbatch --nodelist=node001 env2
Submitted batch job 11836

[joeuser@aun001 memory]$ export INPUT=sinput4
[joeuser@aun001 memory]$ sbatch --nodelist=node001 env2
Submitted batch job 11837
```

Which node are we using? It is in the nodes* file

We have specified the input file here. This is picked up by the script

Force the rest of our jobs to the same node

(If you have a reserved node you can specify it for the first run also.)

Our output files:

```
[joeuser@aun001 memory]$ squeue -u joeuser
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             11836     debug   hybrid  joeuser  R       0:10      1 node001
             11837     debug   hybrid  joeuser  R       0:10      1 node001
             11835     debug   hybrid  joeuser  R       0:24      1 node001
             11834     debug   hybrid  joeuser  R       1:24      1 node001
```

Note different job numbers

```
[joeuser@mio test]$ ls -l fillmemc.sinput*

-rw-rw-r-- 1 joeuser joeuser 2763 Sep 10 20:35 fillmemc.sinput3.output.11836
-rw-rw-r-- 1 joeuser joeuser 2716 Sep 10 20:35 fillmemc.sinput4.output.11837
-rw-rw-r-- 1 joeuser joeuser 2481 Sep 10 20:35 fillmemc.sinput2.output.11835
-rw-rw-r-- 1 joeuser joeuser 2481 Sep 10 20:34 fillmemc.sinput1.output.11834
[joeuser@mio test]$
```

The input file name becomes part of the output file name

Our execution line

```
./$APP  < $INPUT >> $APP.$INPUT.$PBS_JOBID
```

Case 2: Multiple Executables Same Script

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --share
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID

export APP=fillmemc


export OMP_NUM_THREADS=2
for INPUT in  sinput1 sinput2 sinput3 sinput4 ; do

  srun  ./$APP  < $INPUT >> $APP.$INPUT.output.$JOBID &
done
wait
```

We have our application name and input file set as a variable

We launch the application over a list of input files

The wait command "holds" the node until all of your applications are done

Forces the job into the background so we can launch the next

multiwait

Our output files:

Note common job numbers with different input files

```
[joeuser@aun001 memory]$ ls -lt fillmemc*  head -4
-rw-rw-r-- 1 joeuser joeuser    695 Sep 10 20:52 fillmemc.sinput3.output.11839
-rw-rw-r-- 1 joeuser joeuser    695 Sep 10 20:52 fillmemc.sinput1.output.11839
-rw-rw-r-- 1 joeuser joeuser    695 Sep 10 20:52 fillmemc.sinput2.output.11839
-rw-rw-r-- 1 joeuser joeuser    695 Sep 10 20:52 fillmemc.sinput4.output.11839
```

```
srun ./$APP  < $INPUT >> $APP.$INPUT.$JOB &
```

mpiexec/srun and serial applications

- Some versions of mpiexec and srun will work with nonMPI programs
- Creates specified number of copies of the program, all independent

Our Batch file, batch I

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --share
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID

export APP=info.py

  srun  ./$APP
wait
```

info_p is a python program that creates a file based on node name and process id

Running a serial program with mpiexec

export MYPROGRAM=info_p

```
[joeuser@aun001 memory]$ sbatch -p debug serial
Submitted batch job 11841
```

```
[joeuser@aun001 memory]$ cat nodes.11841
node[001-002]
[joeuser@aun001 memory]$ ls -lt node00*
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007602
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007604
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007607
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007606
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007605
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007608
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007609
-rw-rw-r-- 1 joeuser joeuser 39 Sep 10 21:05 node002_00007603
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026256
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026257
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026253
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026255
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026254
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026259
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026260
-rw-rw-r-- 1 joeuser joeuser 40 Sep 10 21:05 node001_00026258
```

```
[joeuser@aun001 memory]$ cat node001_00026256
Python says hello from 26256 on node001
```

Mapping Tasks to Nodes

Need better than default mappings...

- Want to use less than all of the nodes on a node
  - Large memory/task
  - Hybrid MPI/OpenMPI
- Different executables on various cores (MPMD)
- Heterogeneous environment with different core counts

Method for OpenMPI and MVAPICH2

- Same method works for both versions of MPI
- Create a description of your job on the fly from inside your script
- The description is a mapping of programs to cores
- Tell mpiexec/mpirun to use your description to launch your job
- We created a utility script to make it easy

Alternate syntax for mpiexec

- Normally you specify the number of MPI tasks on the mpiexec line
- The alternate syntax is to provide an "appfile"
  - mpiexec -app appfile
  - The appfile is a mapping of executables to nodes

Note: The normal "parallel run" command under slrum is srun. CSM machines AuN and Mio also support mpiexec under slurm. The method discussed here will work on these machines. However, there is also a slurm specific method for doing mappings that will be discussed below. It splits the node list and the application list into two separate files. An easy way to use the slurm specific method is to first create the appfile as discussed here. There is a CSM written utility to split the appfile into the two separate files.

Appfile format

- Collection of lines of the form
  - -host <host name> -np <number of copies to run on host> <program name>
- Specify different application names in your appfile for MPMD
- You can specify a node or program more than once

Examples

These two are equivalent

Appfile Example 1

```
-host compute-1-1 -np 1 myprogram
-host compute-1-1 -np 1 myprogram
-host compute-1-1 -np 1 myprogram
-host compute-1-1 -np 1 myprogram
```

Appfile Example 2

```
-host compute-1-1 -np 4 myprogram
```

Note: You should specify the full path to your program

These two are not equivalent

Appfile Example 3

```
-host compute-1-1 -np 2 aya.out
-host compute-2-3 -np 2 bee.out
```

Appfile Example 3

```
-host compute-1-1 -np 1 aya.out
-host compute-2-3 -np 1 aya.out
-host compute-1-1 -np 2 bee.out
-host compute-2-3 -np 2 bee.out
```

Difficulty and Solution

- Problem:
  - Names of the nodes that you are using are not known until after the job is submitted
  - You need to create the appfile on the fly from within your script

Difficulty and Solution

- Solution:
  - Under PBS the variable $PBS_NODEFILE contains the name of a file that has the list of nodes on which your job will run.
  - Under Slrum the variable SLURM_JOB_NODELIST has a compressed list of nodes.
  - We have created a script "match" which takes a list of nodes and a list of applications to run on those nodes and creates an appfile
  - Located on Mio and AuN at /opt/utility/match

Solution

Given your $PBS_NODEFILE and a list of programs in a file app_list the simplest usage of match is:

```
match $PBS_NODEFILE app_list > appfile
mpiexec --app appfile
```

For mvapich2 replace --app with --configfile

Match notes

- Number of applications that get launched
  - Is equal to the length of the longer of the two lists, the node file list, or the application list
  - If the lists are not the same length then multiple copies will be launched
  - Match also takes an optional replication count, the number copies of an application to run on a node
- Feel free to copy and modify the match script for your own needs

Examples

```
#get a copy of all of our nodes, each node will be
#listed 8 times
cat $PBS_NODEFILE > fulllist

#save a nicely sorted short list of nodes, each node only
#listed one time
sort -u $PBS_NODEFILE > shortlist
```

fulllist

```
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-15.local
compute-8-13.local
compute-8-13.local
compute-8-13.local
compute-8-13.local
compute-8-13.local
compute-8-13.local
compute-8-13.local
compute-8-13.local
```

shortlist

```
compute-8-15.local
compute-8-13.local
```

```
/lustre/home/apps/utility/nsort $PBS_NODEFILE
```

Also works to give a sorted list

Examples

- We have two programs we are going to play with f_ex00 and c_ex00
- We have two program lists that we are going to use
  - oneprogram
    - c_ex00
  - twoprograms
    - c_ex00
    - f_ex00

match fulllist twoprograms > appfile1

```
-host  compute-8-15.local  -np  1  c_ex00
-host  compute-8-15.local  -np  1  f_ex00
-host  compute-8-15.local  -np  1  c_ex00
-host  compute-8-15.local  -np  1  f_ex00
-host  compute-8-15.local  -np  1  c_ex00
-host  compute-8-15.local  -np  1  f_ex00
-host  compute-8-15.local  -np  1  c_ex00
-host  compute-8-15.local  -np  1  f_ex00
-host  compute-8-13.local  -np  1  c_ex00
-host  compute-8-13.local  -np  1  f_ex00
-host  compute-8-13.local  -np  1  c_ex00
-host  compute-8-13.local  -np  1  f_ex00
-host  compute-8-13.local  -np  1  c_ex00
-host  compute-8-13.local  -np  1  f_ex00
-host  compute-8-13.local  -np  1  c_ex00
-host  compute-8-13.local  -np  1  f_ex00
```

match shortlist twoprograms > appfile2

```
-host  compute-8-13.local  -np  1  c_ex00
-host  compute-8-15.local  -np  1  f_ex00
```

match shortlist twoprograms 2 > appfile3

```
-host  compute-8-13.local  -np  2  c_ex00
-host  compute-8-15.local  -np  2  f_ex00
```

match shortlist oneprogram 2 > appfile4

```
-host  compute-8-13.local  -np  2  c_ex00
-host  compute-8-15.local  -np  2  c_ex00
```

This will be useful for hybrid MPI OpenMP

Can take names from command line

match <node list file> -p"list of programs" [<number of copies per node>]

```
match shortlist -p"c_ex01 f_ex01" 1 8
```

Run 1 copy of c_ex01 on the first node in shortlist and 8 copies of f_ex01 on the second node

If you don't specify the number of copies then do 1 per core

Running on Heterogeneous Nodes

- Mixed numbers of cores (8,12, 16)
- Want to use all of the cores
- The number of cores expected on a node is ppn=N
- Could use match with a fixed core count but this might leave some open or over subscribed

```
match shortlist -p"c_ex01" 8 8
```

- If you don't specify the number of copies then you will be given 1 per core

```
match shortlist -p"c_ex01"
```

Slurm Specific mapping

- Slurm has several ways to specify application to node mapping
- mpiexec works on Mio and AuN
- Another way:
  - Node list and application list go in separate files, say hostlist and applist
  - To Run on 12 cores:

```
export SLURM_HOSTFILE=hostlist
srun -n12 --multi-prog applist
```

Slurm Specific mapping

- The hostlist and app list are of the form:

```
HOSTLIST:    APPLIST:
node001      0 helloc
node002      1 hellof
node002      2 hellof
node002      3 hellof
```

- We have the issues creating these files as we do for the mpiexec appfile
- We have a script that converts a mpiexec app file to these separate files

match_split

```
Usage:

/opt/utility/match_split [MATCHFILE applist hostlist]

A post processing script for the CSM utility match.

It takes the mpiexec "appfile" output from match and
creates srun style application list and hostfile list
files for use with the srun option "--multi-prog".

You can pipe "|"  match into this program in which
case /opt/utility/match_split will create the files applist
and hostlist.

Of you can specify the file created by match on the command
line in which case the files created will be of the form
MATCHFILE_applist and MATCHFILE_hostlist.

Finally, you can specify all three files on the command line

/opt/utility/match_split MATCHFILE applist hostlist.

To run a slurm job using these files you do two things:

export SLURM_HOSTFILE=hostlist
srun -n12 --multi-prog applist

where -n12 specifies the total number of MPI tasks to start.
```

match_split

Examples:

```
[joeuser@aun001 mpi]$ cat matchfile
-host  node001  -np  1  helloc
-host  node002  -np  3  hellof
[joeuser@aun001 mpi]$
[joeuser@aun001 mpi]$ /opt/utility/match_split matchfile
[joeuser@aun001 mpi]$ cat matchfile_applist
0 helloc
1 hellof
2 hellof
3 hellof
[joeuser@aun001 mpi]$ cat matchfile_hostlist
node001
node002
node002
node002

export SLURM_HOSTFILE=matchfile_hostlist
srun -n4 --multi-prog matchfile_applist
```

match_split

```
[joeuser@aun001 mpi]$ match shortlist -p"c01 f01" 3 2 | /opt/utility/match_split
[joeuser@aun001 mpi]$
[joeuser@aun001 mpi]$ cat applist
0 c01
1 c01
2 c01
3 f01
4 f01
[joeuser@aun001 mpi]$ cat hostlist
node001
node001
node001
node002
node002

export SLURM_HOSTFILE=hostlist
srun -n5 --multi-prog applist
```

Slurm script for match and match_split

```
#!/bin/bash -x
#SBATCH --job-name="match"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
#SBATCH --ntasks=32
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH –time=00:02:00

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID
#create a shortlist of nodes
/opt/utility/expands $SLURM_JOB_NODELIST | sort -u > shortlist

#run match to create a mpiexec appfile
/opt/utility/match shortlist -p"helloc hellof" 4 8 > appfile

#run the job using mpiexec
mpiexec --app appfile > outone.$JOBID

#run match_split to create a srun applist and hostlist
/opt/utility/match_split appfile applist hostlist

#run the job using srun
export SLURM_HOSTFILE=hostlist
srun -n12 --multi-prog applist > out2.$JOBID
```

First run we use match to create and appfile and run using mpiexec

Then we create separate app list and hostlist files and run using srun

split

Output:

```
[joeuser@aun001 mpi]$ cat appfile
-host  node001  -np  4  helloc
-host  node002  -np  8  hellof
```

```
[joeuser@aun001 mpi]$ cat applist
0 helloc
1 helloc
2 helloc
3 helloc
4 hellof
5 hellof
6 hellof
7 hellof
8 hellof
9 hellof
10 hellof
11 hellof
```

```
[joeuser@aun001 mpi]$ cat hostlist
node001
node001
node001
node001
node002
node002
node002
node002
node002
node002
node002
node002
```

```
[joeuser@aun001 mpi]$ cat out2.11895 | sort
C-> Hello from node001 # 0 of 12
C-> Hello from node001 # 1 of 12
C-> Hello from node001 # 2 of 12
C-> Hello from node001 # 3 of 12
F-> Hello from             node002 #    10 of    12
F-> Hello from             node002 #    11 of    12
F-> Hello from             node002 #     4 of    12
F-> Hello from             node002 #     5 of    12
F-> Hello from             node002 #     6 of    12
F-> Hello from             node002 #     7 of    12
F-> Hello from             node002 #     8 of    12
F-> Hello from             node002 #     9 of    12
```

Creating Directories on the fly
Using Local Disk

A Directory for Each Run

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu

# 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
#MYBASE=$SCRATCH/mc2_tests

# We could create a directoy for our run based on the date/time
export NEW_DIR=`date +%y%m%d%H%M%S`
# But here we create a directoy for our run based on the $JOBID and go there
mkdir -p $MYBASE/$JOBID
cd $MYBASE/$JOBID
odir=`pwd`
export ODIR=`pwd`
# Create a link back to our starting directory
ln -s $SLURM_SUBMIT_DIR submit
cp $SLURM_SUBMIT_DIR/data.tgz .
tar -xzf data.tgz
```

We could create NEW_DIR based on a time stamp, the year, month, day, hour, minute, and second.

or

Based on JOBID

"Copy" our data to the new directory from our staring directory

Local Disk Space

- Most parallel machines have some disk space that is local
  - Can only be seen by tasks on the same nodes
  - Can't be seen from the primary compute node
  - Might be faster than shared space
  - Size? Location?
    - Usually a bad idea to use /tmp
    - On "Rocks" it's /state/partition1
- Usage is up to local policy (we recommend against it)

Using Local disk

- Figure out where it is
- Create directories
- Copy to the new directory
- Compute
- Copy what you want to shared storage
- Clean up after yourself

Here it is...

- Figure out where local disk is
- Create a shared directory where all of the results will be copied in the end
- Get a list of nodes
- Use ssh to create a directory on each node
- Do all the "normal" saves done in other examples
- Go to the new directory (This only happens on master node.)
- Use "match" to create an appfile
- Run the application (All tasks will get launched in the same named directory)
- Use scp to copy the files to shared storage
- Clean up

Set Up

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --ntasks=16
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH --mail-type=ALL
#SBATCH --mail-user=joeuser@mines.edu

# 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

# We could create a directoy for our run based on the date/time
export NEW_DIR=`date +%y%m%d%H%M%S`
# But here we create a directoy for our run based on the $JOBID and go there
mkdir -p $MYBASE/$JOBID
cd $MYBASE/$JOBID
odir=`pwd`
export ODIR=`pwd`
# Create a link back to our starting directory
```

Set Up 2

```
# Create a link back to our starting directory
ln -s $SLURM_SUBMIT_DIR submit
cp $SLURM_SUBMIT_DIR/data.tgz .
tar -xzf data.tgz

if [ -n "$JOBTMP" ] ; then
  echo using $JOBTMP from environment
else
  export JOBTMP=~/scratch/local_sim
fi



#get a list of nodes...
export nlist=`/opt/utility/expands`
# For each node...
for i in $nlist
do
# Create my temporary directory in /scratch on each node
  ssh $i mkdir -p $JOBTMP/$NEW_DIR
# Copy my data
  echo $USER@$i:$JOBTMP/$NEW_DIR
  scp * $USER@$i:$JOBTMP/$NEW_DIR
done

# save a copy of our nodes
/opt/utility/expands > nlist.$JOBID
```


local

Run

```
export APP=$SLURM_SUBMIT_DIR/sinkfile
match $ODIR/flist.$JOBID -p"$APP" > appfile
mpiexec -app appfile  >& screen.$JOBID
```

Clean Up

```
#for each node...
for i in $nlist
do
# Copy files from our local space on each node back to
# my working directory creating a subdirectory for each node.
  mkdir -p $ODIR/$i
  scp -r $USER@$i:$JOBTMP/$NEW_DIR/* $USER@aun.mines.edu:$ODIR/$i
###### or ######
# ssh -r $USER@$i cp -r $JOBTMP/$NEW_DIR/* $SLURM_SUBMIT_DIR/$i


# Remove the temporary directory
ssh $i rm -r $JOBTMP/$NEW_DIR
done
```

Chaining Jobs

Running jobs in sequence

- In theory a batch script can submit another script
- One script creates a second then runs it
- Most systems don't support submission from compute nodes
- Must run from primary compute node
  - ssh mio.mines.edu "cd rundir ; sbatch next_run"
- In most cases it is better to use the batch dependency option

Depend section of the srun man page

-d, --dependency=<dependency_list>
Defer the start of this job until the specified dependencies have been satisfied completed. <dependency_list> is of the form <type:job_id[:job_id][,type:job_id[:job_id]]>. Many jobs can share the same dependency and these jobs may even belong to different users. The value may be changed after job submission using the scontrol command.

after:job_id[:jobid...]
This job can begin execution after the specified jobs have begun execution.

afterany:job_id[:jobid...]
This job can begin execution after the specified jobs have terminated.

afternotok:job_id[:jobid...]
This job can begin execution after the specified jobs have terminated in some failed state (non-zero exit code, node failure, timed out, etc).

afterok:job_id[:jobid...]
This job can begin execution after the specified jobs have successfully executed (ran to completion with an exit code of zero).

expand:job_id
Resources allocated to this job should be used to expand the specified job. The job to expand must share the same QOS (Quality of Service) and partition. Gang scheduling of resources in the partition is also not supported.

singleton
This job can begin execution after any previously launched jobs sharing the same job name and user have terminated.

```
[tkaiser@mio001 examples]$ sbatch old_new
Submitted batch job 4001899
[tkaiser@mio001 examples]$
```

```
srun --dependency=after:123 /tmp/script
srun --dependency=before:234 /tmp/script
```

Auto Chaining Jobs

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=01:00:00
#SBATCH --exclusive
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j
#SBATCH --export=ALL

#----------------------
cd $SLURM_SUBMIT_DIR
module purge
module load StdEnv

# 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

cd $NEW_DIR

# Here we just run the hello world program “phostname"
srun -n 8 /opt/utility/phostname -F -s 10000000 >$SLURM_JOBID.out
```

```
#!/bin/bash
unset OLD_DIR
export NEW_DIR=job1
jid=`sbatch old_new | awk '{print $NF }'`
echo $jid

for job in job2 job3 job4 job5 ; do
  export OLD_DIR=$NEW_DIR
  export NEW_DIR=$job
  jid=`sbatch --dependency=afterok:$jid old_new | awk '{print $NF }'`
  echo $jid
done
```

“chain” runs the script “old_new” 5 times, each time in a new directory and coping previous results

http://geco.mines.edu/files/userguides/techReports/slurmchaining/slurm_errors.html

srun - requesting specific nodes

- srun normally gives you any node
- Can select nodes based on queues
- Can select nodes base on name
- Can select nodes base on features (constraint)
  - slurmnodes -fAvailableFeatures

```
sbatch -p debug

sbatch -p group_name

sbatch --nodelist=node[001-006]

sbatch --constraint=core28
```

Redirection Revisited

```
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=01:00:00
#SBATCH --exclusive
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j
#SBATCH –export=ALL

#----------------------
cd $SLURM_SUBMIT_DIR
module purge
module load StdEnv
###############
# http://compgroups.net/comp.unix.shell/bash-changing-stdout/497180
# set up our redirects of stdout and stderr
                          # 1 and 2 are file descriptors for
                          # stdout and stderr
                          # 3 and 4 are descriptors to logfile
                          # we will use 3 for stdout 4 for stderr
exec 3>>logfile.`date +"%y%m%d%H%M%S"`
                          # anything that goes to 4 will go to 3
                          # which is our file we have created
exec 4>&3
exec 5>&1 6>&2            # save "pointers" to stdin and stdout
exec 1>&3 2>&4            # redirect stdin and stdout to file
###############
# normal commands
                          # this line goes to stdout
echo this is a test from stdout
                          # this line goes to stderr
echo this is a test from stderr >&2
                          # error message goes to stderr
ls file_that_does_not_exist
ls
srun -n 8 /opt/utility/phostname -F -s 10000000 > myout.$SLURM_JOBID
srun -n 8 /opt/utility/phostname -F -s 10000000
###############
exec 1>&5 2>&6            # restore original stdin and stdout
3>&- 4>&-                 # close logfile descriptors
5>&- 6>&-                 # close saved stdin and stdout
```

nodes

A Digression, Not a Batch Script...

```
#!/bin/bash
# $SLURM_NODEFILE defined? (running in batch)
if [ -n "$SLURM_NODELIST" ] ; then
    echo $SLURM_NODELIST
else
# $SLURM_NODELIST not defined
# this implies we are not running in batch
# list all the nodes
  scontrol show nodes -o   | cut -d" " -f1,7 \
                           | sed "s/NodeName=//"
                           | sed “s/CPULoad=//"
  sinfo -a
fi
```

This script prints your nodes if running in batch or all nodes if running on the front end nodes

Slurm array jobs

- Slurm allows array jobs:

Job arrays offer a mechanism for submitting and managing collections of similar jobs quickly and easily. All jobs must have the same initial options (e.g. size, time limit, etc.), however it is possible to change some of these options after the job has begun execution using the command specifying the *JobID* of the array or individual *ArrayJobID*.

Job arrays will have two additional environment variable set. **SLURM_ARRAY_JOB_ID** will be set to the first job ID of the array. **SLURM_ARRAY_TASK_ID** will be set to the job array index value. For example a job submission of this sort:

sbatch --array=1-3 -N1 some_script
will generate a job array containing three jobs. If the sbatch command responds
Submitted batch job 36
then the environment variables will be set as follows:

SLURM_JOBID=36
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=1

SLURM_JOBID=37
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=2

SLURM_JOBID=38
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=3

Slurm array jobs

```
#!/bin/bash -x
#SBATCH --job-name="array"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --share
#SBATCH --export=ALL
#SBATCH --time=00:02:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

# 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`

echo $SLURM_JOB_NODELIST > nodes.$JOBID

export INPUT=sinput${SLURM_ARRAY_TASK_ID}
export APP=fillmemc


/opt/utility/expands $SLURM_JOB_NODELIST >  $APP.$INPUT.nodes.$JOBID
cat $INPUT >  $APP.$INPUT.input.$JOBID
export OMP_NUM_THREADS=2
srun  ./$APP  < $INPUT >> $APP.$INPUT.output.$JOBID.$SLURM_ARRAY_JOB_ID.$SLURM_ARRAY_TASK_ID
```

This script is similar to the last but we use SLURM_ARRAY_TASK_ID and SLURM_ARRAY_JOB_ID to define input and output files

Slurm array jobs

```
[joeuser@aun001 memory]$ sbatch -p debug --nodelist=node002 --array=1-4 array
Submitted batch job 11968


[joeuser@aun001 memory]$ squeue -u joeuser
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          11968_1     debug   hybrid  joeuser  R       0:05      1 node002
          11968_2     debug   hybrid  joeuser  R       0:05      1 node002
          11968_3     debug   hybrid  joeuser  R       0:05      1 node002
          11968_4     debug   hybrid  joeuser  R       0:05      1 node002
```

Here we run 4 jobs on the same node producing:

```
[joeuser@aun001 memory]$ ls -lt | head
total 12768
-rw-rw-r-- 1 joeuser joeuser   1791 Sep 11 14:50 stderr.11968
-rw-rw-r-- 1 joeuser joeuser    803 Sep 11 14:50 stderr.11970
-rw-rw-r-- 1 joeuser joeuser    803 Sep 11 14:50 stderr.11971
-rw-rw-r-- 1 joeuser joeuser    803 Sep 11 14:50 stderr.11969
-rw-rw-r-- 1 joeuser joeuser   2763 Sep 11 14:50 fillmemc.sinput1.output.11968.11968.1
-rw-rw-r-- 1 joeuser joeuser   2763 Sep 11 14:50 fillmemc.sinput4.output.11971.11968.4
-rw-rw-r-- 1 joeuser joeuser   2763 Sep 11 14:50 fillmemc.sinput2.output.11969.11968.2
-rw-rw-r-- 1 joeuser joeuser   2763 Sep 11 14:50 fillmemc.sinput3.output.11970.11968.3
```

With that we say good day