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

Matrix Size N x N

Time (Seconds)
### Sparse Linear Solve on a given number of cores using PETSc - ex16.c

<table>
<thead>
<tr>
<th>Size (M x M)</th>
<th>8</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>26</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>1.7</td>
<td>2.2</td>
<td>2.5</td>
<td>2.7</td>
<td>3.4</td>
<td>4.2</td>
<td>4.6</td>
<td>5.1</td>
</tr>
<tr>
<td>22,500</td>
<td>1.8</td>
<td>2.3</td>
<td>2.6</td>
<td>2.8</td>
<td>3.5</td>
<td>4.3</td>
<td>4.8</td>
<td>5.4</td>
</tr>
<tr>
<td>40,000</td>
<td>2.0</td>
<td>2.3</td>
<td>2.6</td>
<td>2.9</td>
<td>3.6</td>
<td>4.4</td>
<td>4.7</td>
<td>5.3</td>
</tr>
<tr>
<td>62,500</td>
<td>2.0</td>
<td>2.4</td>
<td>2.7</td>
<td>3.0</td>
<td>3.7</td>
<td>4.6</td>
<td>4.9</td>
<td>5.5</td>
</tr>
<tr>
<td>90,000</td>
<td>2.5</td>
<td>2.6</td>
<td>2.9</td>
<td>3.2</td>
<td>3.7</td>
<td>4.7</td>
<td>5.3</td>
<td>5.7</td>
</tr>
<tr>
<td>122,500</td>
<td>2.7</td>
<td>3.0</td>
<td>3.1</td>
<td>3.4</td>
<td>4.0</td>
<td>5.0</td>
<td>5.2</td>
<td>5.8</td>
</tr>
<tr>
<td>160,000</td>
<td>3.5</td>
<td>3.3</td>
<td>3.7</td>
<td>3.6</td>
<td>4.3</td>
<td>5.1</td>
<td>5.5</td>
<td>6.0</td>
</tr>
<tr>
<td>202,500</td>
<td>4.4</td>
<td>4.0</td>
<td>3.9</td>
<td>4.4</td>
<td>4.6</td>
<td>5.5</td>
<td>5.7</td>
<td>6.3</td>
</tr>
<tr>
<td>250,000</td>
<td>5.6</td>
<td>4.8</td>
<td>4.7</td>
<td>4.6</td>
<td>5.2</td>
<td>5.8</td>
<td>6.3</td>
<td>6.9</td>
</tr>
<tr>
<td>302,500</td>
<td>5.8</td>
<td>5.5</td>
<td>5.5</td>
<td>5.4</td>
<td>6.2</td>
<td>6.4</td>
<td>6.8</td>
<td>7.2</td>
</tr>
<tr>
<td>360,000</td>
<td>8.5</td>
<td>7.0</td>
<td>5.3</td>
<td>6.8</td>
<td>6.9</td>
<td>6.8</td>
<td>7.2</td>
<td>8.0</td>
</tr>
<tr>
<td>422,500</td>
<td>9.6</td>
<td>8.0</td>
<td>7.8</td>
<td>7.9</td>
<td>7.5</td>
<td>7.9</td>
<td>7.6</td>
<td>9.0</td>
</tr>
<tr>
<td>490,000</td>
<td>9.2</td>
<td>10.9</td>
<td>9.9</td>
<td>7.9</td>
<td>8.5</td>
<td>9.4</td>
<td>9.5</td>
<td>9.6</td>
</tr>
<tr>
<td>562,500</td>
<td>16.8</td>
<td>13.2</td>
<td>10.7</td>
<td>11.0</td>
<td>10.6</td>
<td>9.4</td>
<td>11.1</td>
<td>10.7</td>
</tr>
<tr>
<td>640,000</td>
<td>19.9</td>
<td>9.6</td>
<td>15.0</td>
<td>13.1</td>
<td>12.0</td>
<td>12.1</td>
<td>12.0</td>
<td>13.0</td>
</tr>
<tr>
<td>722,500</td>
<td>24.3</td>
<td>16.5</td>
<td>17.5</td>
<td>13.0</td>
<td>10.2</td>
<td>12.6</td>
<td>13.3</td>
<td>14.1</td>
</tr>
<tr>
<td>810,000</td>
<td>22.8</td>
<td>18.7</td>
<td>20.5</td>
<td>19.0</td>
<td>14.6</td>
<td>15.3</td>
<td>15.6</td>
<td>14.9</td>
</tr>
<tr>
<td>902,500</td>
<td>17.8</td>
<td>26.7</td>
<td>21.4</td>
<td>23.0</td>
<td>18.8</td>
<td>13.6</td>
<td>17.7</td>
<td>17.8</td>
</tr>
<tr>
<td>1,000,000</td>
<td>17.0</td>
<td>26.8</td>
<td>26.5</td>
<td>26.3</td>
<td>18.0</td>
<td>20.4</td>
<td>19.9</td>
<td>17.2</td>
</tr>
</tbody>
</table>
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]$ wget http://geco.mines.edu/scripts/morescripts.tgz
[joeuser@mio tests]$ tar -xzf morescripts.tgz
[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
```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);
}
```
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
#!/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,mynam);

    MPI_Finalize();
}
Fortran MPI example

```fortran
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

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

```bash
date ; echo “line 2” ; uptime
```

```bash
date

echo “line 2”
date
```
Notes on Commands

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

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

```sh
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

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
## Combing Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>![EXPR]</td>
<td>True if <strong>EXPR</strong> is false.</td>
</tr>
<tr>
<td>[( EXPR )]</td>
<td>Returns the value of <strong>EXPR</strong>. This may be used to override the normal precedence of operators.</td>
</tr>
<tr>
<td>[ EXPR1 -a EXPR2 ]</td>
<td>True if both <strong>EXPR1</strong> and <strong>EXPR2</strong> are true.</td>
</tr>
<tr>
<td>[ EXPR1 -o EXPR2 ]</td>
<td>True if either <strong>EXPR1</strong> or <strong>EXPR2</strong> is true.</td>
</tr>
</tbody>
</table>
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

```bash
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
done
```

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
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

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
## File Tests

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

```bash
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

```bash
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
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

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

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

<table>
<thead>
<tr>
<th>task</th>
<th>thread</th>
<th>node name</th>
<th>first task</th>
<th># on node</th>
<th>core</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>compute176</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0000</td>
<td>0001</td>
<td>compute176</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0000</td>
<td>0002</td>
<td>compute176</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0000</td>
<td>0003</td>
<td>compute176</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0000</td>
<td>compute176</td>
<td>0000</td>
<td>0001</td>
<td>0012</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
<td>compute176</td>
<td>0000</td>
<td>0001</td>
<td>0012</td>
</tr>
<tr>
<td>0001</td>
<td>0002</td>
<td>compute176</td>
<td>0000</td>
<td>0001</td>
<td>0012</td>
</tr>
<tr>
<td>0001</td>
<td>0003</td>
<td>compute176</td>
<td>0000</td>
<td>0001</td>
<td>0012</td>
</tr>
<tr>
<td>0002</td>
<td>0000</td>
<td>compute176</td>
<td>0000</td>
<td>0002</td>
<td>0001</td>
</tr>
<tr>
<td>0002</td>
<td>0001</td>
<td>compute176</td>
<td>0000</td>
<td>0002</td>
<td>0001</td>
</tr>
<tr>
<td>0002</td>
<td>0002</td>
<td>compute176</td>
<td>0000</td>
<td>0002</td>
<td>0001</td>
</tr>
<tr>
<td>0003</td>
<td>0000</td>
<td>compute176</td>
<td>0003</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0003</td>
<td>0001</td>
<td>compute177</td>
<td>0003</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0003</td>
<td>0002</td>
<td>compute177</td>
<td>0003</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0003</td>
<td>0003</td>
<td>compute177</td>
<td>0003</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

This is not what I expected. How do I fix it?
A Simple Slurm Script for a MPI job

```bash
#!/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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/bash</code></td>
<td>This is a bash script</td>
</tr>
<tr>
<td><code>#SBATCH --job-name=&quot;atest&quot;</code></td>
<td>Give our job a name in the scheduler</td>
</tr>
<tr>
<td><code>#SBATCH --nodes=1</code></td>
<td>We want 1 node</td>
</tr>
<tr>
<td><code>#SBATCH --ntasks-per-node=8</code></td>
<td>We expect to run 8 tasks/node</td>
</tr>
<tr>
<td><code>#SBATCH --time=00:02:00</code></td>
<td>We want the node for 2 minutes</td>
</tr>
<tr>
<td><code>#SBATCH --output=stdout</code></td>
<td>Output will go to a file “stdout”</td>
</tr>
<tr>
<td><code>#SBATCH --error=stderr</code></td>
<td>Errors will go to a file “stderr”</td>
</tr>
<tr>
<td><code>#SBATCH --export=ALL</code></td>
<td>Pass current environment to nodes</td>
</tr>
<tr>
<td><code>#SBATCH --mail-type=ALL</code></td>
<td>Send email on abort, begin, end</td>
</tr>
<tr>
<td><code>#SBATCH --mail-user=joeuser@mines.edu</code></td>
<td>Address for email</td>
</tr>
<tr>
<td><code>cd /home/joeuser/examples</code></td>
<td>Go to this directory first</td>
</tr>
<tr>
<td><code>srun -n 8 ./c_ex00</code></td>
<td>Run c_ex00 on 8 cores</td>
</tr>
</tbody>
</table>
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

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

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
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
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
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.#####
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 directory 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
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]$```

![Email inbox screenshot](image)
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 directory 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`

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
#!/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 directory 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.

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.

Save a copy of our input and put the output in its own file.
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]$ cat fillmemc.sinput.nodes.11714 | sort -u
node001

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

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

```bash
#!/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

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
```

Save a list of nodes, first in just a nodes.* file and then a file that contains the input file name.
#!/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 directory from which our job was launched
cd $SLURM_SUBMIT_DIR

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

#export INPUT=sinput
export APP=fillmemc

/export/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.

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

Which node are we using? It is in the mynodes* 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@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]$

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.

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

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]$ 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]$ squeue -u joeuser

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11836</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>0:10</td>
<td>1</td>
<td>node001</td>
</tr>
<tr>
<td>11837</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>0:10</td>
<td>1</td>
<td>node001</td>
</tr>
<tr>
<td>11835</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>0:24</td>
<td>1</td>
<td>node001</td>
</tr>
<tr>
<td>11834</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>1:24</td>
<td>1</td>
<td>node001</td>
</tr>
</tbody>
</table>

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

Note different job numbers

The input file name becomes part of the output file name

./$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 directory 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.
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, batch1

#!/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 directory 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_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 slurm 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

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

Note: You should specify the full path to your program.
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 `$PBS_NODEFILE` contains the name of a file that has the list of nodes on which your job will run.
  - Under Slurm the variable `SLURM_JOB_NODELIST` has a compressed list of nodes.
  - We have created a script "match" which takes a list of nodes and a list of applications to run on those nodes and creates an appfile.
  - Located on Mio and AuN at `/opt/utility/match`
Given your `$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-8-13.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

<table>
<thead>
<tr>
<th>-host</th>
<th>compute-8-15.local</th>
<th>-np</th>
<th>1</th>
<th>c_ex00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-host</td>
<td>compute-8-15.local</td>
<td>-np</td>
<td>1</td>
<td>f_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-15.local</td>
<td>-np</td>
<td>1</td>
<td>c_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-15.local</td>
<td>-np</td>
<td>1</td>
<td>f_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-15.local</td>
<td>-np</td>
<td>1</td>
<td>c_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-15.local</td>
<td>-np</td>
<td>1</td>
<td>f_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-15.local</td>
<td>-np</td>
<td>1</td>
<td>c_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-15.local</td>
<td>-np</td>
<td>1</td>
<td>f_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-13.local</td>
<td>-np</td>
<td>1</td>
<td>c_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-13.local</td>
<td>-np</td>
<td>1</td>
<td>f_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-13.local</td>
<td>-np</td>
<td>1</td>
<td>c_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-13.local</td>
<td>-np</td>
<td>1</td>
<td>f_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-13.local</td>
<td>-np</td>
<td>1</td>
<td>c_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-13.local</td>
<td>-np</td>
<td>1</td>
<td>f_ex00</td>
</tr>
<tr>
<td>-host</td>
<td>compute-8-13.local</td>
<td>-np</td>
<td>1</td>
<td>c_ex00</td>
</tr>
</tbody>
</table>
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:**
  - node001
  - node002
  - node002
  - node002

  **APPLIST:**
  - 0 helloc
  - 1 hellof
  - 2 hellof
  - 3 hellof

- We have the issues creating these files as we do for the mpiexec appfile

- We have a script that converts a mpiexec appfile 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 -n 4 --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 directory 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
#!/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 directory 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 directory for our run based on the date/time
export NEW_DIR=`date +%y%m%d%H%M%S`
# But here we create a directory 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
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 directory from which our job was launched
cd $SLURM_SUBMIT_DIR

# Create a short JOBID based 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 directory for our run based on the date/time
export NEW_DIR=`date +%y%m%d%H%M%S`
# But here we create a directory 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
# 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
Run

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

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

"chain" runs the script "old_new" 5 times, each time in a new directory and coping previous results
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
#!/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.$SLURM_JOBID
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
#!/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
```
#!/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 directory 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

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>Nodelist (Reason)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11968_1</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>0:05</td>
<td>1</td>
<td>node002</td>
</tr>
<tr>
<td>11968_2</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>0:05</td>
<td>1</td>
<td>node002</td>
</tr>
<tr>
<td>11968_3</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>0:05</td>
<td>1</td>
<td>node002</td>
</tr>
<tr>
<td>11968_4</td>
<td>debug</td>
<td>hybrid</td>
<td>joeuser</td>
<td>R</td>
<td>0:05</td>
<td>1</td>
<td>node002</td>
</tr>
</tbody>
</table>

Here we run 4 jobs on the same node producing:

[joeuser@aun001 memory]$ ls -lt | head

```bash
-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