JASMIN Workshop: Exercise 09: Compile and run parallel Fortran code on LOTUS

Scenario

Parallel computing is the use of two or more processors to solve a large problem size. There are different forms of parallelism, Shared Memory parallelism (threading) e.g. OpenMP and Distributed Memory parallelism which uses the Message Passing Interface (MPI). This exercise will demonstrate how to compile and test a parallel MPI Fortran code on LOTUS.

The Fortran code generates two vectors `X(n)` and `Y(n)` of `n=2**10` elements and then calculates a vector `Z(n)` as `Z(i)= a * X(i) + Y(i)`. The code outputs the maximum value of the vector elements Z(i), `maxval(abs(Z))`

A very important part of this workflow is that there are **2 separate steps**:

1. **Compilation** of code:
    a. On a single LOTUS host
    b. Run interactively
2. **Execution** of compiled code:
    a. On the required number of LOTUS hosts
    b. Run as batch job(s)

There are a limited number of licences available for compilers so please adhere to this 2-step approach.

Objectives

- To be able to compile and test a Fortran serial and parallel codes on LOTUS host interactively
- To learn about the special LSF submission options to require compute resources for parallel MPI codes
- To be able to submit an MPI job to the batch system LSF

JASMIN resources

- GWS workshop
- Compiler Intel and MPI library
- LOTUS queues `par-single, par-multi`
- Scientific analysis server `jasmin-sci3` or `jasmin-sci6`
- Fortran source codes serial, MPI are provided
  `/group_workspaces/jasmin2/workshop/exercises/ex09/src`

Local resources

- SSH client to login to JASMIN

Instructions

1. Launch two terminals, start the ssh-agent and add JASMIN private key (skip if already done) then login to JASMIN on each terminal
2. On terminal 1, SSH to the scientific analysis server `jasmin-sci[3,6].ceda.ac.uk`

3. On terminal 2, SSH to a different scientific analysis server
4. On terminal 1, invoke a pseudo-interactive session on LOTUS with a single core `bsub -n 1 -q short-serial -Is /bin/bash`
5. On terminal 2, run the LSF command `bjobs`
6. On terminal 1, load the Intel compiler module and check the module is loaded `module load intel/17.0`
7. On terminal 1, copy the Fortran source code from the exercise directory to the current working directory `cp /group_workspaces/jasmin2/workshop/exercises/ex09/src/* ./`
8. On terminal 1, compile the Fortran serial code `ifort axpySerial.f90 -o  axpySerial.exe`
   Note: if a large workshop class of JASMIN users try to compile the Fortran code at the same time, the **compilation is guaranteed to fail and require multiple retries**. This is due to the limited number of licence tokens available for Intel compilers on JASMIN.
9. On terminal 1, execute the Fortran serial binary `./axpySerial.exe`
10. On terminal 1, exit the pseudo-interactive session on LOTUS `exit`
11. On terminal 1, Invoke a pseudo-interactive session on LOTUS host with 4 cores `bsub -n 4  -q par-single -Is /bin/bash`
12. On terminal 2, run the LSF command `bjobs`
13. On terminal 1, load Intel compiler and compile the Fortran MPI code `mpif90 axpyMPI.f90 -o axpyMPI.exe`
14. On terminal 1, execute the Fortran MPI binary using 4 CPU cores `mpirun.lotus -np 4 axpyMPI.exe`
15. Exit the interactive session on LOTUS `exit`
16. On terminal 1, launch a text editor to prepare the jobscript `axpyMPI.bsub` to submit binary MPI compiled earlier e.g. `axpyMPI.bsub`

```
#!/bin/bash
#1 Specify the LOTUS queue: -q <queue-name>
#BSUB

#2 standard job output and error files: -o <filename>_%J.out, -e
<filename>_%J.err Note: the special character %J in the filename will be
replaced by the job ID assigned by LSF when the job is submitted
#BSUB
#BSUB

#3 Assign a name to the job: -J "<job_name>"
#BSUB

#4 Set maximum runtime or walltime: -W HH:MM
#BSUB

#5 Specify the number of CPU cores: -n <number-of-cores>
#BSUB
#6 Distribute cores on LOTUS hosts -R "span[ptile=<number-of-cores-per-host>]"
#BSUB

#7 Executable
mpirun.lotus <Executable-compiled-interactively-on-LOTUS>
```

17. Submit the job to the LSF batch system and note the job ID `bsub < axpyMPI.bsub`
18. On terminal 2, monitor your job using LSF command `bjobs`
19. List the job output and error file for jobID

20. Inspect the resources used by the MPI job in the job standard output, e.g use the command `less <output-file>`
21. Logout

Review

By completing this exercise you will be able to compile and test a serial and parallel MPI Fortran code interactively on LOTUS. You will be able to use the special submission flags to submit an MPI job to LSF. You will be able to use compilers via the module environment. MPI message passing interface is a library to facilitate data sharing and communication between CPU cores -often called ranks- as each rank access its own data space.

Alternative approaches and best practice

- **The Platform MPI** library is the only supported MPI library on LOTUS. It provides at least 10% speedup compared to mpich-gm for cluster applications, is scalable to higher core counts than other MPI libraries and , more importantly, supports the full range of interconnects from one library.
- It is necessary to use **`mpirun.lotus`** to execute MPI parallel codes. It is a wrapper around the native Platform MPI to ensure the use of the special LSF launch mechanism (`blaunch`) and forces the MPI communications to run over the private MPI network.
- LOTUS **`par-single`** and **`par-multi`** are dedicated queues for MPI and OpenMP parallel code
- Run the MPI code on a single core before scaling up to many cores
- **`/work/scratch`** can be used for MPI and parallel IO jobs. Please clean up the work scratch area
- Keep your source code in your home directory which is backed up
- There is a limited number of licences available for compilers, so please do not submit many jobs to compile the same code.
- Run the MPI code on the same CPU model used to compile the MPI source code **`-m <host-group-name>`** https://help.jasmin.ac.uk/article/211-lotus-hardware
- Testing serial and parallel code using LOTUS queues is illustrated at https://help.jasmin.ac.uk/article/193-compile-and-test-jobs

Cheat sheet for Exercise 09: Compile and run parallel Fortran code on LOTUS

1. Launch two terminals. Start the ssh-agent and load your SSH private key(skip if already done). Login to JASMIN

```
eval $(ssh-agent -s)
ssh-add ~/.ssh/<your-private-key-file>
ssh -A <username>@jasmin-login1.ceda.ac.uk
```

2. On terminal 1, SSH to the scientific analysis server `jasmin-sci[3,6]`

```
ssh <username>@jasmin-sci6.ceda.ac.uk
```

3. On terminal 2, SSH to a different scientific analysis server

```
jasmin-login1 ~]$ ssh <username>@jasmin-sci1.ceda.ac.uk
```

4. On terminal 1, Invoke a pseudo-interactive session on LOTUS

```
jasmin-sci6 ~]$ bsub -n 1 -q short-serial -Is /bin/bash
Job <4311717> is submitted to queue <workshop>.
<<Waiting for dispatch ...>>
<<Starting on host586.jc.rl.ac.uk>>
[fchami@host586 ~]
```

5. On terminal 2, run the LSF command

```
jasmin-sci1 ~]$ bjobs
JOBID   USER    STAT  QUEUE     FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
4311717 freddy  RUN   workshop  host586.jc. host584.jc. /bin/bash  Jun 25 16:55
```

6. On the pseudo-interactive session -Terminal 1, load the Intel compiler modulefile and check the module is loaded

```
[freddy@host586 ~] module load intel/17.0
[freddy@host586 ~] module li
 Currently Loaded Modulefiles:
  1) lsfmodules/9.1      3) intel/cce/17.0.0     5) intel/17.0
  2) lotus-mpi/8.2       4) intel/fce/17.0.0
```

7. Copy the source code from the exercise directory to the current working directory

```
[freddy@host586 ~] cp /group_workspaces/jasmin2/workshop/exercises/ex09/src/* ./
```

8. Compile the Fortran sequential code

```
[freddy@host586 ~] ifort axpySerial.f90 -o  axpySerial.exe
```

9. Execute the Fortran sequential binary

```
[freddy@host586 ~] ./axpySerial.exe
```

10. Exit the interactive session

```
[freddy@host586 ~] exit
```

11. Invoke a pseudo-interactive session on LOTUS host with 4 cores

```
jasmin-sci6 ~]$ bsub -n 4  -q par-single -Is /bin/bash
Job <4538517> is submitted to queue <workshop>.
<<Waiting for dispatch ...>>
<<Starting on host583.jc.rl.ac.uk>>
[freddy@host583
```

12. From jasmin-login1.ceda.ac.uk SSH to another scientific analysis server and run the LSF command

```
jasmin-sci1 ~]$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
4538517 freddy  RUN   workshop   host492.jc. 4*host583.j /bin/bash Jun 25 17:00
```

13. Load Intel compiler and compile the Fortran MPI code

```
[freddy@host583 ~] module load intel/17.0
[freddy@host583 ~] mpif90 axpyMPI.f90 -o axpyMPI.exe
```

14. Execute the Fortran MPI binary using 4 CPU cores

```
[freddy@host583 ~] mpirun.lotus -np 4 axpyMPI.exe
```

15. Exit the interactive session on LOTUS

```
[freddy@host583 ~] exit
```

16. Launch a text editor to prepare the jobscript file or use the template files. Note: use the binary output of the compiled Fortran MPI code earlier in the pseudo-interactive session

```
#!/bin/bash
#specify the LOTUS queue -q <queue-name>
#BSUB -q par-multi

# standard job output and error files
#BSUB -o axpyMPI.%J.out
#BSUB -e axpyMPI.%J.err

# Assign a name to the job  -J "<job_name>"
#BSUB -J "axpyMPI"

#Set maximum runtime or walltime -W HH:MM
#BSUB -W 00:10

# Specify Memory requirement XXX in units of MB if  > 8000 MB and set the
# memory control flag to terminate the job if it exceeds reserved memory XXX
## BSUB -R "rusage[mem=XXX]" -M XXX

# Specify the number of CPU cores -n <number-of-cores>
#BSUB -n 4
# Distribute cores on LOTUS hosts -R "span[ptile=<number-of-cores-per-host>]"
#BSUB -R "span[ptile=2]"

# Executable
mpirun.lotus axpyMPI.exe
```

17. Submit your job to LOTUS

```
jasmin-sci6 ~]$ bsub < axpyMPI.bsub
Job <4535533> is submitted to queue <workshop>.
```

18. Monitor your job

```
jasmin-sci1 ~]$ bjobs
JOBID   USER    STAT  QUEUE       FROM_HOST    EXEC_HOST    JOB_NAME   SUBMIT_TIME
4535533 freddy RUN short-serial   jasmin-sci1 2*host586.j axpyMPI   Jun 2515:06
                                             2*host585.jc.rl.ac.uk
```

19. List the job output and error file for jobID `4535533`

```
jasmin-sci1 ~]$ ls -l axpyMPI.4535533*
-rw-r--r-- 1 freddy users    0 Jun 25 15:06 axpyMPI.4535533.err
-rw-r--r-- 1 freddy users 2641 Jun 25 15:06 axpyMPI.4535533.out
```

20. Inspect the resources used by the MPI job in the job standard output, e.g use the command `less <output-file>` . The standard error file is empty. Hence the job successfully completed.

```
Resource usage summary:

    CPU time :                                   1.11 sec.
    Max Memory :                                 4 MB
    Average Memory :                             4.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   20 MB
    Max Processes :                              1
    Max Threads :                                1
    Run time :                                   6 sec.
```

21. Logout