TaiYi Supercomputer
User Manual

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Date: Nov-2018

Qiwang Chen • HPC Application Engineer • chenqw3@lenovo.com

Lenovo Shenzhen Global Technology Inc. •

Table of Contents

Table of Contents (Continued)

Introduction

This user's guide for the “**TaiYi**” cluster is intended to provide the minimum amount of information needed by a new user of this system. As such, it assumes that the user is familiar with many of the standard features of supercomputing as the Unix operating system.

Here you can find most of the information you need to use our computing resources and the technical documentation about the machine. Please read carefully this document and if any doubt arises do not hesitate to contact us (Getting help).

Systems Overview

"**TaiYi**" is a supercomputer based on Intel Xeon Gold processors from the Skylake generation. It is a Lenovo system composed of SD530 Compute Racks, an Intel Omni-Path high performance network interconnect and running RedHat Linux Enterprise Server as operating system. Its current Linpack Rmax Performance is 1.67 Petafops.

TaiYi consists four type of block:

- General Purpose: 2-sockets node
- GPU nodes: 2-sockets with 2x NVIDIA V100 GPU
- Big Memory nodes: 9-sockets with 6TB memory
- FPGA nodes: 2-sockets with 8 Intel A10 FPGA
- KNM nodes: Intel Xeon Phi Processor codenamed Knights Mill

This **general-purpose** block consists of 13 racks housing 815 nodes with a grand total of 32,600 processor cores and 156 Terabytes of main memory. Compute nodes are equipped with:

- 2-sockets Intel Xeon Gold 6148 CPU with 20 cores each @2.40GHz for a total of 40 cores per node
- L1d cache 32K; L1i cache 32K; L2 cache 1024k; L3 cache 27.5MB
- 192GB of main memory, 4.8 GB/core
- 100Gbit/s Intel Omni-Path HFI silicon 100 Series PCI-E adapter
- 10Gbit RJ-45 Ethernet(operation as 1Gbit)
- 240GB local SSD available as temporary storage during jobs

This GPU block consists of 4 nodes gpu nodes are equipped with:

- 2-sockets Intel Xeon Gold 6148 CPU with 20 cores each @2.40GHz for a total of 40 cores per node
- L1d cache 32K; L1i cache 32K; L2 cache 1024k; L3 cache 27.5MB
- 384GB of main memory, 9.6 GB/core
- 2 NVIDIA Tesla V100 GPU with 5120 NVIDIA CUDA Cores and 16GB GPU memory
- 100Gbit/s Intel Omni-Path HFI silicon 100 Series PCI-E adapter
- 1Gbit RJ-45 Ethernet
- 240GB local SSD for local operation system, and extra 3x 1.92TB SSD available as temporary storage during jobs

The 2 super nodes are equipped with:

- 8-sockets Intel Xeon Platinum 8160 CPU with 24 cores each @2.10GHz for a total of 192 cores per node
- L1d cache 32K; L1i cache 32K; L2 cache 1024k; L3 cache 33MB
- 6TB of main memory, 32GB/core

- 100Gbit/s Intel Omni-Path HFI silicon 100 Series PCI-E adapter
- 1Gbit RJ-45 Ethernet
- 240GB local SSD for local operation system, and extra 2x 3.84TB SSD available as temporary storage during jobs

The 2 FPGA nodes gpu nodes are equipped with:

- 2-sockets Intel Xeon Gold 6148 CPU with 20 cores each @2.40GHz for a total of 40 cores per node
- 192GB of main memory, 4.8 GB/core
- **8x flyslice FA510Q FPGA**
- 100Gbit/s Intel Omni-Path HFI silicon 100 Series PCI-E adapter
- 1Gbit RJ-45 Ethernet
- 1.92TB local SSD available as temporary storage during jobs

The 4 Intel HNS7200APRL KNM node are equipped with:

- 1-sockets Intel Xeon Phi with 72 cores each @1.50GHz, 4-threads per core for a total of 288 threads per node
- 192GB of main memory
- 100Gbit/s Intel Omni-Path HFI silicon 100 Series PCI-E adapter
- 1Gbit RJ-45 Ethernet
- 3.84TB local SSD available as temporary storage during jobs

The processors except KNM support well-known vectorization instructions such as SSE, AVX, AVX256 up to AVX512

Compilation for the architecture

To generate code that is optimized for the target architecture and the supported features such as SSE, AVX, AVX256, AVX512 instruction sets you will have to use the corresponding compile flags. For compilations of MPI applications an MPIinstallation need to be loaded in your session as well. For example Intel MPI via:

```
$ module load mpi/intel/2018.4
```

Intel Compilers

The latest Intel compilers provides the best possible optimizations for the Xeon Platinum/Gold architecture. By default, when staring a new session on the system the basic modules for the Intel suite should be automatically loaded. That is the compilers (intel/2018.4), the Intel MPI software stack (mpi/intel/2018.4) and the math kernel libraries MKL (within intel/2018.4) in their latest versions. We highly recommended linking against Intel MKL where supported to achieve the best performance results.

To separately load the Intel compilers please use:

```
$ module load intel/2018.4
```

The corresponding optimization flags for icc are **CFLAGS="-xCORE-AVX512 –mtune=skylake"**, As the login nodes are of the exact same architecture as the compute node you can use the flag **–xHOST** which enables all possible optimization available on the compile host.

Intel Compiler Licenses

For reasons of licensing we recommend compiling using either **login01/login02/login03/login04/gpu01**, We currently have node locked licenses installed that allow user compilations in the machines. If user try to compile in the rest of the compute nodes an error message like the one below will appear.

Below is icc/ifort error message:

```
Intel(R) Parallel Studio XE 2018 Update 4 for Linux*
Copyright (C) 2009-2018 Intel Corporation. All rights reserved.
[root@c01n01 ~]# icc

Error: A license for Comp-CL could not be obtained (-1,359,2).

Is your license file in the right location and readable?
The location of your license file should be specified via
the $INTEL_LICENSE_FILE environment variable.

License file(s) used were (in this order):
**  1.  /share/intel/2018u4/licenses
**  2.  /opt/intel/licenses
**  3.  /root/intel/licenses
**  4.  /share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/licenses
**  5.  /share/intel/2018u4/clck/2018.3/licenses
**  6.  /Users/Shared/Library/Application Support/Intel/Licenses
**  7.
/share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/bin/intel64/../../Licenses
**  8.  /root/Licenses
**  9.  /share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/bin/intel64/*.lic

Please refer http://software.intel.com/sites/support/ for more information..

icc: error #10052: could not checkout FLEXlm license
```

GCC

The GCC provided by the system is version 4.8.5. for better support of new hardware features we recommend to use the latest version that can be loaded via the provided modules. Currently the latest version available in TaiYi is GCC 8.2.0.

To load GCC 8.2.0, please use:

```
$ module load gcc/8.2.0
```

The corresponding flags are **CLAGS=”-march=skylake-avx512”**

Login Nodes

You can connect to **TaiYI** using four public login nodes. Please note that only incoming connections are allowed in the whole cluster. The logins are:

Hostname	IP Address
Login01	172.18.6.175
Login02	172.18.6.176
Login03	172.18.6.177
Login04	172.18.6.178

Note: ssh to compute nodes is denied. Users must submit jobs using IBM Spectrum LSF.

When login first time, system will generate public/private rsa key.

```
No public/private RSA keypair found.
Generating public/private rsa key pair.
Created directory '/work/test/.ssh'.
Your identification has been saved in /work/chenqw/.ssh/id_rsa.
Your public key has been saved in /work/test/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:lKmhnDUn937ju2C81/Cb0HwcoEJ29yzsnZ36G3iDowQ test@login02.hpc.sustc
The key's randomart image is:
+---[RSA 2048]----+
|         o       |
|      = *o . o   |
|   . + Oo.. + +  |
|    + . SE.. o + |
|         oo oo=.=|
|          =.+OoB+|
|         ..=oo*oo|
|          .o++++.|
+----[SHA256]-----+
```

You can delete ~/.ssh directory to force re-generate public/private key pair.

Password Management

In order to change the password, you have to use **yppasswd**, the **yppasswd** command change your password in the NIS database.

When you enter the **yppasswd** command on the command line, the system prompts you to enter the old password. When you do this, the system prompts you to enter the new password. The password you enter can be as small as four characters long if you use a mixture of uppercase and lowercase characters.

If you enter the old password incorrectly, you have to enter the new password before the system will give you an error message. The system requires both passwords because the update protocol sends them to the server at the same time. The server catches the error and notifies you that you entered the old password incorrectly.

To change a user's NIS password, enter:

```
yppasswd Joe
```

This example demonstrates how to change the NIS password for the user named Joe. The system prompts you to enter Joe's old password and then his new password.

Transfering files

There are two ways to copy files from/to the cluster:

- Direct **scp/sftp** to the login nodes
- Using grapical ssh client, such as winscp, Xmanager xftp, SecureFX

On a Windows system, most of the secure shell clients come with a tool to make secure copies or secure ftp, there are several tools that accomplish the requirements, please refer to the application help page, where you will find the most common ones and examples of use.

IMPORTANT: It is your responsibility as a user of our facilities to backup all your critical data.

Filesystem

Each user has several areas of disk space for storing files. These areas may have size or time limits, please read carefully all this section to know about the policy of usage of each of these filesystems. There are 4 different types of storage available inside a node:

- Local SSD Disk: Is the filesystem where the operation system insides, provide directory **/tmp** for node local scratch. The amount of space within the **/tmp** is about 200GB
- GPFS filesystems: GPFS is a distributed networked filesystem which can be accessed from all the nodes. There are 3 GPFS filesystems inside a node: **/share**, **/work**, **/scratch**

Local SSD Disk

Every node has a local solid state (SSD) hard drive that can be used as a local scratch space to store temporary files during executions of one of your jobs. This space is /tmp, the amount of space within the /tmp filesystem is about 200GB, All data stored in these local hard drives at the compute nodes will not be available from login nodes and other compute nodes. This directory should automatically be cleaned after the job finished. All data stored in /tmp will never be backup.

GPFS Filesystem

The IBM General Parallel File System (GPFS) is a high-performance shared-disk file system providing fast, reliable data access from all nodes of the cluster to a global filesystem, GPFS allows parallel applications simultaneous access to a set of files (even a single file) from any node that has the GPFS file system mounted while providing a high level of control over all file system operations. In addition, GPFS can read or write large blocks of data in a single I/O operation, thereby minimizing overhead.

There are the GPFS filesytems available in the machines from cluster nodes:

- **/share**: Over this filesystem will reside the applications and libraries that have already been installed on the machine. Take a look at the directories to know the applications available for general use.

- **/work**: This is the home directories of all the users, and when you log in you start in your home directory by default. Every user will have their own home directory to store own developed sources and their personal data. A default quota will be enforced on all users or groups to limit the amount of data stored there. Also, it is highly discouraged to run jobs from this filesystem. Please run your jobs on /scratch instead.

- **/scratch**: Each user will have a directory over /scratch. Its intended use is to store temporary files of your jobs during their execution. There is no default quota on this filesystem, all data should be automatically cleaned after the job finishes. Data that is not part of an active job will be cleaned regularly.

- **/data**: In addition the home directory, **only on login nodes**, This space is intended to store user important data. This directory will have limited backup.

Quotas

The quotas are the amount of storage available for a user or a group. You can picture it as a small disk readily available to you. A default value is applied to all users and groups and cannot be outgrown.

You can inspect your quota anytime you want using the following command from inside each filesytem:

```
$ mmlsquota –g groupname --block-size auto
```

```
$ mmlsquota –u username --block-size auto
```

Example mmlsquota output:

```
[root@login01 ~]# mmlsquota -g test --block-size auto

Disk quotas for group test (gid 1000):
                         Block Limits                                    |     File Limits
Filesystem type         blocks      quota      limit   in_doubt    grace |    files   quota
limit in_doubt    grace  Remarks
data       GRP         no limits

                         Block Limits                                    |     File Limits
Filesystem type         blocks      quota      limit   in_doubt    grace |    files   quota
limit in_doubt    grace  Remarks
work       GRP            256K       500G       600G          0     none |       23       0
0        0     none
```

If you need more disk space in any other of the GPFS filesystems, the responsible for your project has to make a request for the extra space needed, specifying the requested space and the reasons why it is needed. For more information or requests you can Contact Us

Default quota are enforced on /work, /data:

- **/work: 500GB per group**

- **/data: 2.0TB per group**

Runing Jobs

IBM Spectrum LSF is the utility used for batch processing support, so all jobs must run through it. This section provides information for getting started with job execution at the cluster.

> **IMPORTANT Note:** For best performance, all jobs should request 40 cores per block. For LSF scripts, you should use –R “span[ptile=40]” and total number of cores can be divisible by 40

Queues

There are several queues present in the machines and different users may access different queues, all queues have different limits in amount of cores for the jobs and duration.

The standard configuration and limits of the queues are the following:

Queue Name	Maximum # of nodes	Maximum wall clock
medium	805	72 hours
short	805	24 hours
smp	2	12 hours
ser	10	24 hours
gpu	4	24 hours
debug	10	1 hours

By default, all jobs will be dispatched to short and medium.

LSF Introduction

The IBM Spectrum LSF ("LSF", short for load sharing facility) software is industry-leading enterprise-class software. LSF distributes work across existing heterogeneous IT resources to create a shared, scalable, and fault-tolerant infrastructure that delivers faster, more reliable workload performance and reduces cost. LSF balances load and allocates resources, and provides access to those resources. An LSF cluster manages resources, accepts and schedules workload, and monitors all events. Users and administrators can access LSF by a command-line interface, API, etc.

On TaiYI, LSF is the batch system that provides job management. Jobs written in other batch system formats must be translated to LSF in order to be used on TaiYi. The Batch Translation Guide offers some assistance for translating between batch systems.

The LSF job script

The LSF takes care of assigning to the user jobs the requested resources so that many jobs can be run simultaneously without interfering with each other, and scheduling the execution of the different requests. The LSF needs some user-provided information to be able to do a good job, that's why the user is required to provide a job script (also called batch file) with the specification of:

- the resources requested, and the job constraints;
- specific queue specification (more details later);
- all what is needed for a working execution-environment.

By resources we mean the number of processors/cores/nodes, the amount of memory needed (total and/or per process), and eventually some specific features (special hardware, for example), and so on.

Job constraints are typically time constraints, i.e. for how much time you are reserving the resources. Notice that there are limitations, both for the maximum number of cores, jobs, and time.

Then you have to provide a functioning execution environment: remember that your application will be run when scheduled by the LSF, so the application must be able to run without user intervention (sometimes called unattended execution). This means that the correct specification of the executables must be provided, with all the necessary input files, and a correct specification of the environment (for example libraries, modules…).

Notice that the LSF reserves the resources for your job (and run it when there are enough, avoiding conflicts with other users' jobs) based on your request. It is therefore important that the resources requested correspond to what really is needed by your application.

All these information must be written in a text file, following a simple syntax that will be shown later.

Once you have this text file (the job script, let us call it job.lsf), you must submit it by typing in a terminal the command:

```
$ bsub < job.lsf
```

Then you can check the status of your submission issuing the command:

```
$ bjobs
```

In a job file, a script directive precedes resource specification options. For each batch system, this directive is different. On TaiYi (LSF) this directive is **#BSUB**.

For every line of resource specifications, this directive must be the first text of the line, and all specifications must come before any executable lines. An example of a resource specification is given below:

```
#BSUB -J MyJobName  #Set the job name to "MyJobName"
```

Note: Comments in a job file also begin with a # but LSF recognizes **#BSUB** as a directive.

A list of the most commonly used and important options for these job files are given in the following:

Basic Job Specifications

Several of the most important options are described below. These basic options are typically all that is needed to run a job on TaiYi.

Basic LSF Job Specifications			
Specification	**Option**	**Example**	**Example-Purpose**
Job Name	-J [SomeText]	-J MyJob1	Set the job name to "MyJob1"
Shell	-L [Shell]	-L /bin/bash	Uses the bash shell to initialize the job's execution environment.
Wall Clock Limit	-W [hh:mm]	-W 24:15	Set wall clock limit to 24 hour 15 min
Core count	-n ##	-n 400	Assigns 400 job cores.
Cores per node	-R "span[ptile=##]"	-R "span[ptile=40]"	Request 40 cores per node.
Memory Per Core	-M [##]	-M 2GB	Sets the per process memory limit to 2GB.
Memory Per Core	-R "rusage[mem=[##]]"	-R "rusage[mem=2GB]"	Schedules job on nodes that have at least 2GBs available per core.
Combined stdout and stderr	-o [OutputName].%j	-o stdout1.%j	Collect stdout/err in stdout.[JobID]

Optional Job Specifications

A variety of optional specifications is available to customize your job. The table below lists the specifications, which are most useful for users.

Optional LSF Job Specifications			
Specification	**Option**	**Example**	**Example-Purpose**
Set Allocation	-P ######	-P projectA	Set allocation to charge to Project projectA
Specify Queue	-q [queue]	-q short	Request only nodes in short subset.
Exclusive Node Usage	-x		Assigns a whole node exclusively for the job.
Specific node type	-R "select[gpu|phi]"	-R "select[gpu]"	Requests a node with a GPU to be used for the job.

Environment Variables

All the nodes enlisted for the execution of a job carry most of the environment variables the login process created: **HOME, SCRATCH, PWD, PATH, USER,** etc. In addition, LSF defines new ones in the environment of an executing job. Below is a list of most commonly used environment variables.

LSF Environment Variables		
Variable	**Usage**	**Description**
Job ID	$LSB_JOBID	Batch job ID assigned by LSF.
Job Name	$LSB_JOBNAME	The name of the Job.
Queue	$LSB_QUEUE	The name of the queue the job is dispatched from.

Error File	$LSB_ERRORFILE	Name of the error file specified with a bsub -e.
Submit Directory	$LSB_SUBCWD	The directory the job was submitted from.
Hosts I	$LSB_HOSTS	The list of nodes that are used to run the batch job, repeated according to ptile value. *The character limit of LSB_HOSTS variable is 4096.
Hosts II	$LSB_MCPU_HOSTS	The list of nodes and the specified or default ptile value per node to run the batch job.
Host file	$LSB_DJOB_HOSTFILE	The hostfile containing the list of nodes that are used to run the batch job.

Note: To see all relevant LSF environment variables for a job, add the following line to the **executable section** of a job file and submit that job. All the variables will be printed in the output file.

```
env | grep LSB
```

Executable Commands

After the resource, specification section of a job file comes the executable section. This executable section contains all the necessary UNIX, Linux, and program commands that will be run in the job.

Some commands that may go in this section include, but are not limited to:

- Changing directories
- Loading, unloading, and listing modules
- Launching software

Note:

- Remember that the UNIX is case sensitive and so uppercase and lowercase letters have different meanings, even in the job script specifications.
- Avoid using special characters, letters and spaces in the name of files, directories, and the job name. The script could be interpreted by the scheduler in unexpected ways.
- The default behaviour in LSF is to append the content of the file. So if you don't use unique names (like in the example), subsequent runs will add to the same file. To change the default behaviour to overwrite, use the flags -oo and -eo.
- If you do not specify the error file but only the output one, the standard output and standard error streams are merged and saved into the output file.
- The filename after the -o, -e flag can be a full path. If it is the name of an existing directory (ending with "/"), then LSF will automatically save as file with a name <jobid>.out.
- If you don't specify to save the output file somewhere, the content of the standard output and error streams will be appended to the report that is sent after completion (up to a system-predefined maximum size).

Job Submission

Once you have your job file ready, it is time to submit your job. You can submit your job to LSF with the following command:

`bsub < job.lsf` (**Note: Symbol "<" is mandatory between bsub and script**)

```
Job <44951> is submitted to default queue .
```

The integer number between the brackets is the job-id.

Manage LSF Jobs

After a job has been submitted, you may want to check on its progress or cancel it. Most of a job's characteristics can be seen by invoking `bjobs -l <jobID>`. LSF captures and reports the exit code of the job script (bsub jobs) as well as the signal that caused the job's termination when a signal caused a job's termination.

A job's record remains in LSF's memory for **5 minutes** after it completes. `bjobs -l <jobID>`will return "`Job <jobID> is not found`" for a job that completed more than 5 minutes ago. At that point, one must invoke the **bhist** command to retrieve the job's record from the LSF database.

If a user's job is in the pending state waiting to be scheduled, the user can prevent the job from being scheduled by invoking the `bstop <jobID>` command to place the job into a PSUSP state. Jobs in the held state do not accrue any job priority based on queue wait time. Once the user is ready for the job to become a candidate for scheduling once again, they can release the job using the `bresume <jobID>` command.

Pending jobs can be cancelled (withdrawn from the queue) using the **bkill** command (`bkill <jobID>`). The **bkill** command can also be used to terminate a running job. The default behavior is to issue the job a SIGTERM, wait 30 seconds, and if processes from the job continue to run, issue a SIGKILL command.

The -s option of the bkill command (`bkill -s <signal> <jobID>`) allows the user to issue any signal to a running job.

Below is a list of the most used job monitoring and control commands for jobs:

LSF Job Monitoring and Control Commands		
Function	**Command**	**Example**
Submit a job	bsub < [script_file]	bsub < MyJob.LSF
Cancel/Kill a job	bkill [Job_ID]	bkill 101204
Check summary status of a single job	bjobs [job_id]	bjobs 101204
Check summary status of all jobs for a user	bjobs -u [user_name]	bjobs -u User1
Check detailed status of a single job	bjobs -l [job_id]	bjobs -l 101204
Modify job submission options	bmod [bsub_options] [job_id]	bmod -W 2:00 101204
Holding a batch job	bstop [jobid]	bstop 101204
Releasing a batch job	bresume [jobid]	bresume 101204

For more information on any of the commands above, please see their respective *man* pages.

```
$ man [command]
```

LSF Job States

The basic job states are:

- **PEND** - the job is in the queue, waiting to be scheduled
- **PSUSP** - the job was submitted, but was put in the suspended state (ineligible to run)
- **RUN** - the job has been granted an allocation. If it's a batch job, the batch script has been run
- **DONE** - the job has completed successfully

For the complete list, please refer:
https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_admin/job_state_lsf.html

A pending job can remain pending for a number of reasons:

- Dependency - the pending job is waiting for another job to complete
- Priority - the job is not high enough in the queue
- Resources - the job is high in the queue, but there are not enough resources to satisfy the job's request

For the complete list, see the "Pending jobs" section under the About jobs states:
http://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_admin/job_state_lsf.html

LSF Job Templates

On the section below, you can find some basic lsf job examples:

MPI Jobs under LSF

```
#!/bin/bash
#
#BSUB -J MPIJob             ### set the job Name
#BSUB -q short              ### specify queue
#BSUB -n 40                 ### ask for number of cores (default: 1)
#BSUB –R “span[ptile=40]”   ### ask for 40 cores per node
#BSUB -W 10:00              ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out      ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err      ### -o and -e mean append, -oo and -eo mean overwrite

# here follow the commands you want to execute

# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load intel/2018.4
module load mpi/intel/2018.4

### This uses the LSB_DJOB_NUMPROC to assign all the cores reserved
### This is a very basic syntax. For more complex examples, see the documentation
mpirun -np $LSB_DJOB_NUMPROC ./MPI_program
```

when using system-wide intelmpi/openmpi, you can omit the `-np $LSB_DJOB_NUMPROC` command, because the program gets the information about the total number of cores automatically, If you use different MPI library, it may be necessary to specify explicitly the numberof MPI processes on the command line in this way.

OpenMP Jobs under LSF

This is an example script for running an OpenMP application under LSF.

There are at least two items you have to specify:

- The number of cores you request for the job.
- The name and options of your program.

```
#!/bin/bash
#
#BSUB -J OpenMPjob          ### set the job Name
#BSUB -q short              ### specify queue
#BSUB -n 40                 ### ask for number of cores (default: 1)
#BSUB -R "span[hosts=1]"    ### specify that the cores MUST BE on a single host!
#BSUB -W 16:00              ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out      ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err      ### -o and -e mean append, -oo and -eo mean overwrite

# set OMP_NUM_THREADS _and_ export!
OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
export OMP_NUM_THREADS

# Program_name_and_options
```

```
./openmp_program [options]
```

If you make use of special environment variables for your OpenMP program, remember to put them in your script (use the same syntax as the `OMP_NUM_THREADS` line in the script).

Hybrid MPI/OpenMP jobs under LSF

MPI and OpenMP can be combined to write programs that exploit the possibility given by multi-core SMP machines, and the possibility of using the Message Passing Interface based communications between nodes on the cluster.

Special care must be taken to write the job script, for this kind of problem.

Typically, one would want to run a different MPI process on each node, and then a certain number of openMP threads on each node.

An example script is the following:

```
#!/bin/bash
#
#BSUB –J Hybrid_MPI_OpenMP  ### set the job Name
#BSUB -q short              ### specify queue
#BSUB -n 400                ### ask for number of cores (default: 1)
#BSUB –R “span[ptile=40]”   ### ask for 40 cores per node
#BSUB -W 10:00              ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out      ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err      ### -o and -e mean append, -oo and -eo mean overwrite

# here follow the commands you want to execute

# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load intel/2018.4
module load mpi/intel/2018.4

# -- OpenMP environment variables --
Num=$(echo $LSB_SUB_RES_REQ | sed -e 's/span\[ptile=/ /g' -e 's/\]/ /g')
OMP_NUM_THREADS=$Num
export OMP_NUM_THREADS

mpirun -npernode 1 ./mpi_bybrid_program
```

This program asks for a total of 400 cores, 40 on each of 10 nodes. On each of these nodes it will run 1 MPI process, that will in turn use 40 OpenMP threads. We want the threads to run on different nodes, and not all on a single node (this may not be important in general, but in same case it can be crucial).

Run the program through the MPI wrapper, specifying that only 1 MPI process per node should be started:

`mpirun -npernode 1 your_hybrid_program`

If you need to specify other OpenMP special environment variable, follow the same syntax as for `OMP_NUM_THREADS`

Job Dependencies under LSF

There may be situations where there are obvious dependencies between jobs, and one cannot start before another has finished (for example because it is dependent on the results of the previous jobs).

LSF provides a dependency feature that can be used for this purpose. One can therefore specify in the batch script of a job, that this job has a dependency on another one. This is done with the syntax

```
#BSUB -w "done(<jobid>)"
```

This will ensure that the current job is started only after the job with the corresponding jobid is completed successfully. Some of the most common options are

- **done** : job state done
- **ended** : job state exit or done
- **exit** : job state is EXIT and eventually with a specific exit code exit(<jobid>,<exit-code>)
- **started** : job state DONE, EXIT, USUSP, SSUSP or RUN

In case of a job array, the array-id can also be specified with the syntax

```
#BSUB -w "done(<jobid>[job-index])"
```

To know the jobs that a job depends on, type

```
bjdepinfo <jobid>
```

To show the jobs that depend on a job, use the -c oprion

```
bjdepinfo -c <jobid>
```

Using GPUs under LSF10

Currently TaiYi have 4 nodes with NVIDIA V100 GPU, some application should be able to utilize the GPU, here is an example job script:

```
#!/bin/bash
#
#BSUB –J GPUJob             ### set the job Name
#BSUB -q gpu                ### specify queue
#BSUB -n 40                 ### ask for number of cores (default: 1)
#BSUB –R “span[ptile=40]”   ### ask for 40 cores per node
#BSUB -W 10:00              ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out      ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err      ### -o and -e mean append, -oo and -eo mean overwrite

# here follow the commands you want to execute

# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load cuda/10.0
module load mpi/openmpi/3.1.2_gcc

mpirun ./gpu_program
```

Recommended Settings for Large Jobs

For jobs larger than 2000 cores (50+ nodes), the following MPI settings are recommended to reduce the MPI startup time:

```
export I_MPI_HYDRA_BOOTSTRAP=lsf
export I_MPI_HYDRA_BRANCH_COUNT=XXX
export I_MPI_LSF_USE_COLLECTIVE_LAUNCH=1
```

The XXX number should match the number of nodes (#Nodes = #Cores / #CoresPerNode) that your job will request.

Batch Job Translation Guide

This section contains helpful informations for the purposes of moving from SLURM/PBS/Torque to LSF

User Commands

The table below contains commands that would be run from the login nodes of the cluster for job control and monitoring purposes.

User Commands	LSF	Slurm	PBS/Torque
Job submission	bsub [script_file]	sbatch [script_file]	qsub [script_file]
Job deletion	bkill [job_id]	scancel [job_id]	qdel [job_id]
Job status (by job)	bjobs [job_id]	squeue --job [job_id]	qstat [job_id]
Job status (by user)	bjobs -u [user_name]	squeue -u [user_name]	qstat -u [user_name]
Queue list	bqueues	squeue	qstat -Q

Environment Variables

The table below lists environment variables that can be useful inside job files.

Environment Variables	LSF	Slurm	PBS/Torque
Job ID	$LSB_JOBID	$SLURM_JOBID	$PBS_JOBID
Submit Directory	$LSB_SUBCWD	$SLURM_SUBMIT_DIR	$PBS_O_WORKDIR

Job Specifications

The table below lists various directives that set characteristics and resources requirements for jobs.

Job Specification	LSF	Slurm	PBS/Torque
Script directive	#BSUB	#SBATCH	#PBS
Node Count	N/A (Calculated from: CPUs/CPUs_per_node)	-N [min[-max]]	-l nodes=[count]
CPUs Per	-R "span[ptile=count]"	--ntasks-per-node=[count]	-l ppn=[count]

Node			
CPU Count	-n [count]	-n [count]	N/A (Calculated from: CPUs_per_node * Nodes)
Wall Clock Limit	-W [hh:mm]	-t [min] or -t[days-hh:mm:ss]	-l walltime=[hh:mm:ss]
Memory Per Core	-M [mm] AND -R "rusage[mem=mm]" * Where mm is in MB	--mem-per-cpu=mem[M/G/T]	-l mem=[mm] * Where mm is in MB
Standard Output File	-o [file_name]	-o [file_name]	-o [file_name]
Standard Error File	-e [file_name]	-e [file_name]	-e [file_name]
Combine stdout/err	(use -o without -e)	(use -o without -e)	-j oe (both to stdout) OR -j eo (both to stderr)
Event Notification	-B and/or -N * For Begin and/or End	--mail-type=[ALL/END] * See 'man sbatch' for all types	-m [a/b/e] * For Abort, Begin, and/or End
Email Address	-u [address]	--mail-user=[address]	-M [address]
Job Name	-J [name]	--job-name=[name]	-N [name]
Account to charge	-P [account]	-account=[account]	-l billto=[account]
Queue	-q [queue]	-q [queue]	-q [queue]

Software Environment

All software and numerical libraries available at the cluster can be found at **/share**. If you need something that is not there please contact us to get it installed.

- /share/apps: applications such as vasp, openmpi, cae, bio, etc.
- /share/base: basic libraries, gcc, etc.
- /share/intel: Intel cluster tools, Intel C/C++/Fortran, MKL, Intel MPI, Vtune, etc.

Compilers

In the cluster you can find these C/C++ compiles:

- icc/icpc: Intel C/C++ Compilers
- gcc/g++: GNU Compilers for C/C++
- ifort: Intel Fortran Compiler
- gfortran: GNU Compiler for Fortran

To find out what version available , you can use the following command:

```
$ module avail intel
----- /share/base/modulefiles/compilers ----------------------
intel/2017.8 intel/2018.3 intel/2018.4
```

check gcc version:

```
$ module avail gcc
----- /share/base/modulefiles/compilers ----------------------
gcc/8.2.0(default)
```

Distributed Memory Parallelism

To compile MPI programs it is recommended to use the following handy wrappers: `mpicc`, `mpicxx` for C and C++ source code, `mpif77/mpif90` for Fortran source code. You need to choose the Parallel environment first. These wrappers will include all the necessary libraries to build MPI applications without having to specify all the details by hand.

For Intel MPI library, we are able to use `mpiicc/mpiicpc/mpiifort` wrappers, which combine intel mpi and intel compilers.

Shared Memory Parallelism

OpenMP directives are fully supported by the Intel C and C++ compilers. To use it, the flag `-qopenmp` must be added to the compile line.

You can also mix MPI + OPENMP code using `-qopenmp` with the mpi wrappers mentioned above.

Basic compiler flags(Intel compiler)

This section introduce some of the most common compiler flags. These flags are accepted by all compilers (icc/icpc/ifort) with some notable exceptions. For a full description of all the compiler flags please consult the appropriate man pages.

Flag	Description
-help [category]	Displays all available compiler options or category of compiler options categories.
-o <file>	Specifies the name for an output file. For an executable, name of output file will be <file> instead of a.out
-c	Only compile the file, linking phase will be skipped
-L <dir>	Tells the linker to search for libraries in directory <dir> ahead of the standard library directories.
-l<name>	Tells the linker to search for library named lib**name**.so or lib**name**.a

Optimization flags(Intel compiler)

The default optimization level for Intel compilers is -O2 (which enables optimizations like inlining, constant/copy propagation, loop unrolling,peephole optimizations, etc). The table below shows some addional commonly used optimization flags that can be used to improve run time.

Flag	Description
-O3	Performs -O2 optimizations and enables more aggressive loop transformations.
-xHost	Tells the compiler to generate vector instructions for the highest instruction set available on the host machine.
-fast	Convenience flag. In linux this is shortcut for -ipo, -O3, -no-prec-div, -static, and -xHost
-ip	Perform inter-procedural optimization within the same file
-ipo	Perform inter-procedural optimization between files
-parallel	enable automatic parallelization by the compiler (very conservative)
-opt-report=[n]	generate optimization report. n represent the level of detail (0 ..3, 3 being most detailed)
-vec-report[=n]	generate vectorization report. n represents the level of detail (0..7 , 7 being most detailed)

NOTE: there is no guarantee that using a combination of the flags above will provide additional speedup compared to -O2. In some rare cases (e.g. floating point imprecision) using flags like -fast might result in code that might produce incorrect results.

Debugging flags (Intel compiler)

The table below shows some compiler flags that can be useful in debugging your program.

Flag	Description
-g	Produces symbolic debug information in the object file.
-warn	Specifies diagnostic messages to be issued by the compiler.
-traceback	Tells the compiler to generate extra information in the object file to provide source file traceback information when a severe error occurs at run time.
-check	Checks for certain conditions at run time (e.g. uninitialized variables, array bounds). Note, since the resulting code includes additional run time checks it may affect run time significantly. THIS IS AN IFORT ONLY FLAG
-fpe0	throw exception for invalid, overflow, divide by zero. THIS IS AN IFORT ONLY FLAG

Flags affecting floating point operations (Intel compiler)

Some optimization might affect how floating point arithmetic is performed. This might result in round off errors in certain cases. The table below shows a number of flags to instruct the compiler how to deal with floating point operations:

Flag	Description
-fp-model precise	disable optimizations that are not value safe on floating point data (See man page for other options)
-fltconsistency	enables improved floating-point consistency. This might slightly reduce execution speed.
-fp-speculation=strict	tells the compiler to disable speculation on floating-point operations (See man page for other options)

Modules Environment

The Environment Modules package (http://modules.sourceforge.net/) provides a dynamic modifcation of a user's environment via modulefiles. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically, in a clean fashion. All popular shells are supported, including bash, ksh, zsh, sh, csh, tcsh, as well as some scripting languages such as perl.

Installed software packages are divided into serval categories:

- Compilers: Compiler suites available for the system (intel, gcc, . . .)
- Parallel: openmpi/mvapich/intelmpi, etc
- Tools: useful tools which can be used at any time (pdsh, darshan, . . .)
- Applications: High Performance Computers programs (VASP, . . .)
- Libraries: Those are tipycally loaded at a compilation time, they load into the environment the correct compiler and linker flags (FFTW, LAPACK, . . .)

To find out what software packages are available under the Modules system use one of the following commands:

```
$ module avail [packageName]
```

In order to use some software on our clusters, the module for the software must be loaded first. To load the module for a software package, use the following command:

```
$ module load packageName/version
```

To find out what packages/modules you have loaded on your current session, use the following command:

```
$ module list
```

To remove one module from your current session, use the following command:

```
$ module unload [packageName]
```

To remove ALL modules from your current session, use the following command:

```
$ module purge
```

Note: It is very important to remove all modules when switching between software and compilers. Mixing modules and versions is NOT a good idea and can cause many problems.