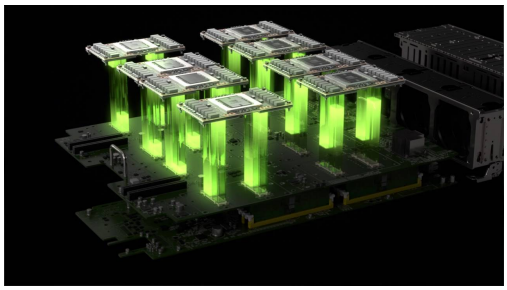
JADE Documentation

Release

Mozhgan K. Chimeh

May 07, 2020

Contents



This is the documentation for the Joint Academic Data science Endeavour (JADE) facility.

JADE is a UK Tier-2 resource, funded by EPSRC, owned by the University of Oxford and hosted at the Hartree Centre. The hardware was supplied and integrated by ATOS Bull.

A consortium of eight UK universities, led by the University of Oxford, has been awarded £3 million by the Engineering and Physical Sciences Research Council (EPSRC) to establish a new computing facility known as the Joint Academic Data science Endeavour (JADE). This forms part of a combined investment of £20m by EPSRC in the UK's regional Tier 2 high-performance computing facilities, which aim to bridge the gap between institutional and national resources.

JADE is unique amongst the Tier 2 centres in being designed for the needs of machine learning and related data science applications. There has been huge growth in machine learning in the last 5 years, and this is the first national facility to support this rapid development, with the university partners including the world-leading machine learning groups in Oxford, Edinburgh, KCL, QMUL, Sheffield and UCL.

The system design exploits the capabilities of NVIDIA's DGX-1 Deep Learning System which has eight of its newest Tesla V100 GPUs tightly coupled by its high-speed NVlink interconnect. NVIDIA has clearly established itself as the leader in massively-parallel computing for deep neural networks, and the DGX-1 runs optimized versions of many standard machine learning software packages such as Caffe, TensorFlow, Theano and Torch.

This system design is also ideal for a large number of molecular dynamics applications and so JADE will also provide a powerful resource for molecular dynamics researchers at Bristol, Edinburgh, Oxford and Southampton.

CHAPTER 1

JADE Hardware

JADE hardware consists of:

- 22 DGX-1 Nodes, each with 8 Nvidia V100 GPUs
- 2 Head nodes
- Mellanox EDR networking
- Over 1PB of Seagate ClusterStor storage

1.1 Using the JADE Facility

The JADE facility consists of 2 head nodes and 22 NVIDIA DGX-1 servers, each with 8 GPUs and 40 CPU cores.

Accounts

Users get accounts on the system by following the directions in the **Getting an account** section, on the left.

New users must provide a public SSH key and are given a user ID on JADE. Using this ID, they will then be able to login to one of the head nodes by using an SSH command like

```
ssh -l account_name jade.hartree.stfc.ac.uk
```

Further details are in the section **Connecting to the cluster using SSH**.

Software

The software packages already installed on JADE comes in two kinds: *standard applications* (primarily Molecular Dynamics) and *containerised applications* (various Machine Learning applications in particular). These are described further in the **Software on JADE** section on the left.

The *module* system is used to control your working environment and the particular version of software which you want to use; details are given in the section **The 'module' tool** on the left.

If not using the installed software, you are also welcome to build your own applications. Codes can be built on the head node, and a number of compilers and MPI library stacks are available via the modules.

Running applications

Applications can only be run on the compute nodes by submitting jobs to the Slurm batch queuing system. Examples of Slurm submission scripts are given in the relevant sections for each of the main software packages.

It is also possible to obtain an interactive session through Slurm on one of the compute nodes. This is usually only for code development purposes; submitting batch jobs is the standard way of working.

Storage

The global file system is accessible from both the head nodes and the compute nodes. Any files written during the job execution on the compute nodes will be found on the file system after the job has completed.

For information on your disk space usage and remaining quota, use the following command

```
getquota
```

There is also access to local disc space on each compute node, but this access only possible during a Slurm job and once the job is completed the local disc data is removed automatically. In machine learning applications, for example, this local disc space (provided by fast SSD) may be useful as a staging point for very large training sets.

1.1.1 Getting an account

As a regular user, getting started involves 3 steps:

1) Apply for a Hartree SAFE account

This is a web account which will show you which projects you belong to, and the accounts which you have in them.

Before applying for a SAFE account, you should first have an SSH key-pair, and be ready to provide your public key as part of the SAFE registration process. If you need help generating an SSH key-pair, contact your local IT support staff.

Once you have your public SSH key ready, apply for your SAFE account by going here: https://um.hartree.stfc.ac.uk/hartree/login.jsp and providing all of the required information.

When your account has been approved, you will receive an email giving your initial password. When you login for the first time you will be asked to change it to a new one.

2) Apply for a JADE project account

Once your SAFE account is established, login to it and click on "Request Join Project".

From the drop-down list select the appropriate **project**, enter the **signup code** which you should have been given by the project PI or manager, and then click "Request".

The approval process goes through several steps:

1. approval by the PI or project manager – once this is done the SAFE status changes to Pending
2. initial account setup – once this is done the SAFE status changes to Active
3. completion of account setup – once this is done you will get an email confirming you are all set, and your SAFE account will have full details on your new project account

This process shouldn't take more than 2 working days. If it takes more than that, check whether the PI or project manager is aware that you have applied, and therefore your application needs their approval through the SAFE system.

If your SAFE userid is xyz, and your project suffix is abc, then your project account username will be xyz-abc and you will login to JADE using the command:

ssh -l xyz-abc jade.hartree.stfc.ac.uk

Note that some users may belong to more than one project, in which case they will have different account usernames for each project, and all of them will be listed on their SAFE web account.

Each project account will have a separate file structure, and separate quotas for GPU time, filestore and other system resources.

Note also that JADE has multiple front-end systems, and because of this some SSH software operating under stringent security settings might give warnings about possible man-in-the-middle attacks because of apparent changes in machine settings. This is a known issue and is being addressed, but in the meantime these warnings can be safely ignored.

3) Apply for a Hartree ServiceNow account

This is a web account used for reporting any operational issues with JADE.

To obtain an account follow the directions here: http://community.hartree.stfc.ac.uk/wiki/site/admin/servicenow.html

Note the guidance which explains that the first time you try to login you will not have a password so you need to click on the link which says "reset your password here".

Due to a problem with synchronising userids between ServiceNow and JADE, it is possible that ServiceNow may say that your email address is not recognised. If this happens, please send an email to hartree@stfc.ac.uk and ask them to add you to the ServiceNow database.

1.1.2 Information for PIs and Project Managers

Creating a Project

To access the JADE HPC facility, please see the information here: Get access to JADE HPC facilities

Getting an Account

After the project proposal has been approved, a SAFE account must be created which allows the management of **users** and **projects**. At registration, the PI provides an institutional email address which will be used as their SAFE login ID (please note, emails such as gmail or hotmail will not be accepted). An SSH Key is also required at this stage, and instructions on how to generate and upload this are provided in SAFE User Guide.

Once a project has been set up by Hartree staff, the PI can define associated project managers and then the PI and project managers are able to define groups (in a tree hierarchy), accept people into the project and allocate them to groups.

Projects and Groups on JADE

At the top-level there are **projects** with a PI, and within a **project** there are **groups** with resources such as GPU time and disk space. A **group must be created** in order to accept normal users in to the project.

The simplest approach is to create a single group for the Project for all users, but additional groups can be created if necessary.

To create a group:

1. From the main portal of SAFE, click the *Administer* button next to the name of the project that you manage.
2. Click the *Project Group Administration* button, then click *Add New*.

3. Type in a name and description and click *Create*.

Project Signup Code

The **project signup code is required** so that users can request to join your project.

To set a project signup code:

1. From the main portal of SAFE, click the *Administer* button next to the name of the project that you manage.
2. Click the *Update* button.
3. In the **Password** field, set the desired **project signup code** and press *Update*.

Adding Users to the Project

After creating a group, users can be added to a project.

To add users:

1. Provide your users with the **project name**, **project signup code** (see above section) and signup instruction for regular users at: http://jade-hpc.readthedocs.io/en/latest/jade/getting-account.html
2. Once a user has requested to join a project, there will be a "New Project Management Requests" box. Click the *Process* button and *Accept* (or *Reject*) the new member.
3. The PI can now add the member to the a **group** previously created.
4. Click on *Administer* for the Project then choose *Project Group Administration*.
5. Click on the *Group name*, then on *Add Account*, and then select any available user from a list (including the PI).
6. There is now a manual process, which may take up to 24 hours, after which the new Project member will be notified of their new userid and invited to log on to the Hartree systems for the first time.
7. The new project member will have an *Active* status in the group once the process is completed.

Note: The PI does NOT have to 'Request Join Project' as he/she is automatically a Project member. They must however, add themselves to a Group.

Project groups and shared file system area

Each Project group maps to a Unix group, and each Project group member's home file system is set up under a directory group structure. The example below starts from a user's home directory and shows that all other members of the Project group are assigned home directories as "peer" directories.

```
-bash-4.1$ pwd
/gpfs/home/training/jpf03/jpf26-jpf03
-bash-4.1$ cd ..
-bash-4.1$ ls
afg27-jpf03  bbl28-jpf03  cxe72-jpf03  dxd46-jpf03  hvs09-jpf03  jjb63-jpf03  jxm09-
↪jpf03  mkk76-jpf03  phw57-jpf03  rrr25-jpf03  rxw47-jpf03  sxl18-jpf03
ajd95-jpf03  bwm51-jpf03  cxl10-jpf03  dxp21-jpf03  hxo76-jpf03  jkj47-jpf03  kxm85-
↪jpf03  mxm86-jpf03  pxj86-jpf03  rrs70-jpf03  sca58-jpf03  tcn16-jpf03
axa59-jpf03  bxp59-jpf03  djc87-jpf03  fxb73-jpf03  ivk29-jpf03  jpf26-jpf03  lim17-
↪jpf03  nxt14-jpf03  rja87-jpf03  rwt21-jpf03  shared       txc61-jpf03
axw52-jpf03  bxv09-jpf03  dwn60-jpf03  gxx38-jpf03  jds89-jpf03  jrh19-jpf03  ltc84-
↪jpf03  pag51-jpf03  rjb98-jpf03  rxl87-jpf03  sls56-jpf03  vvt17-jpf03
```

Important to some, please note that for each Project group there is a "shared" directory which can be reached at

```
../shared
```

from each user's home directory. Every member of the Project group is able to read and write to this shared directory, so it can be used for common files and applications for the Project.

Once a Project has Finished

It is Hartree Centre policy that, after the agreed date of completion of a Project, all data will be made read-only and will then remain retrievable for 3 months. During this period, users are able to login to retrieve their data, but will be unable to run jobs. After 3 months have elapsed, all login access associated with the Project will be terminated, and all data owned by the Project will be deleted.

1.1.3 Connecting to the cluster using SSH

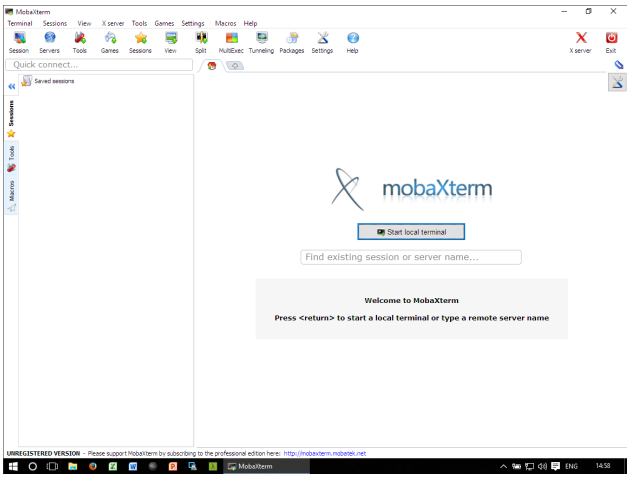
To log onto the JADE cluster you must use SSH, which is a common way of remotely logging in to computers running the Linux operating system.

To do this, you need to have a SSH *client* program installed on your machine. macOS and Linux come with a command-line (text-only) SSH client pre-installed. On Windows there are various graphical SSH clients you can use, including *MobaXTerm*.

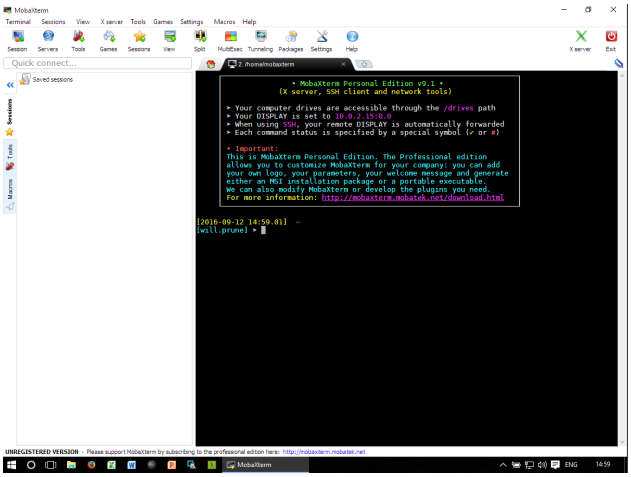
SSH client software on Windows

Download and install the *Installer edition* of mobaxterm.

After starting MobaXterm you should see something like this:



Click *Start local terminal* and if you see something like the following then please continue to *Establishing a SSH connection*.



Running commands from a terminal (from the command-line) may initially be unfamiliar to Windows users but this is the recommended approach for running commands on ShARC and Iceberg as it is the idiomatic way of interfacing with the Linux clusters.

SSH client software on Mac OS/X and Linux

Linux and macOS (OS X) both typically come with a command-line SSH client pre-installed.

If you are using macOS and want to be able to run graphical applications on the clusters then you need to install the latest version of the XQuartz *X Windows server*.

Open a terminal (e.g. *Gnome Terminal* on Linux or *Terminal* on macOS) and then go to *Establishing a SSH connection*.

Establishing a SSH connection

Once you have a terminal open, run the following command to log into one of the JADE front-end nodes:

```
ssh -l $USER jade.hartree.stfc.ac.uk
```

Here you need to replace `$USER` with your username (e.g. `te1st-test`)

Note: **macOS users**: if this fails then:

- Check that your XQuartz is up to date then try again *or*
- Try again with `-Y` instead of `-X`

Note: JADE has multiple front-end systems, and because of this some SSH software operating under stringent security settings might give warnings about possible man-in-the-middle attacks because of apparent changes in machine settings. This is a known issue and is being addressed, but in the meantime these warnings can be safely ignored.

Note: When you login to a cluster you reach one of two login nodes. You **should not** run applications on the login nodes. Running `srun` gives you an interactive terminal on one of the many worker nodes in the cluster.

1.1.4 The `module` tool

Introduction

The Linux operating system makes extensive use of the *working environment*, which is a collection of individual environment variables. An environment variable is a named object in the Linux shell that contains information used by one or more applications; two of the most used such variables are `$HOME`, which defines a user's home directory name, and `$PATH`, which represents a list paths to different executables. A large number of environment variables are already defined when a Linux shell is open but the environment can be customised, either by defining new environment variables relevant to certain applications or by modifying existing ones (e.g. adding a new path to `$PATH`).

`module` is a Software Environment Management tool, which is used to manage the working environment in preparation for running the applications installed on JADE. By loading the module for a certain installed application, the environment variables that are relevant for that application are automatically defined or modified.

Useful commands

The module utility is invoked by the command `module`. This command must be followed by an instruction of what action is required and by the details of a pre-defined module.

The utility displays a help menu by doing:

```
module help
```

The utility displays the available modules by issuing the command:

```
module avail
```

or displays only the information related to a certain software package, *e.g.*:

```
module avail pgi
```

The avail instruction displays all the versions available for the installed applications, and shows which version is pre-defined as being the default. A software package is loaded with the load or the add instructions, *e.g.*:

```
module load pgi
```

If no version is specified, the default version of the software is loaded. Specific versions, other than the default can be loaded by specifying the version, *e.g.*:

```
module load pgi/2017
```

The modules that are already loaded by users in a session are displayed with the command:

```
module list
```

A module can be "unloaded" with the unload or rm instructions, *e.g.*:

```
module unload pgi
module load pgi/2017
```

Lastly, all modules loaded in a session can be "unloaded" with a single command::

```
module purge
```

Best practices

`module` can be used to modify the environment after login, *e.g.* to load the Portland compilers in order to build an application. However, most frequent usage will be to load an already built application, and the best way to do this is from within the submission script. For example, assuming a job uses the NAMD molecular dynamics package, the submission script contains:

```
module purge
module load NAMD
```

1.1.5 The Slurm Scheduler

Introduction

Running software on the JADE system is accomplished with batch jobs, *i.e.* in an unattended, non-interactive manner. Typically a user logs in to the JADE login nodes, prepares a job script and submits it to the job queue.

Jobs on JADE are managed by the Slurm batch system , which is in charge of:

- allocating the computer resources requested for the job,
- running the job and
- reporting the outcome of the execution back to the user.

Running a job involves, at the minimum, the following steps

- preparing a submission script and
- submitting the job to execution.

This guide describes basic job submission and monitoring for Slurm. The generic topics in the guide are:

- the main Slurm commands,
- preparing a submission script,
- submitting a job to the queue,
- monitoring a job execution,
- deleting a job from the queue and
- important environment variables.

Additionally, the following topics specific to JADE are covered (*under construction*)

- slurm partitions,
- using fast local storage and
- controlling affinity.

Commands

The table below gives a short description of the Slurm commands that are likely to be useful to most users.

Command	Description
`sacct`	report job accounting information about active or completed jobs
`sbatch`	submit a job script for later execution (the script typically contains one or more `srun` commands to launch parallel tasks)
`scancel`	cancel a pending or running job
`sinfo`	reports the state of partitions and nodes managed by Slurm (it has a variety of filtering, sorting, and formatting options)
`squeue`	reports the state of jobs (it has a variety of filtering, sorting, and formatting options), by default, reports the running jobs in priority order followed by the pending jobs in priority order

All Slurm commands have extensive help through their man pages *e.g.*:

```
man sbatch
```

shows you the help pages for the `sbatch` command.

In addition to the above commands, the table below gives two more commands that can be used in special cases, *e.g.* to obtain an interactive session, such as used in the Machine Learning examples. The commands are

Command	Description
`salloc`	allocate resources for a job in real time (typically used to allocate resources and spawn a shell, in which the srun command is used to launch parallel tasks)
`srun`	used to submit a job for execution in real time

N.B. `srun` can be used to launch application into execution from within submission scripts. The success of this in the case of MPI distributed applications depends on the MPI software stack having been build with support for PMI (Process Management Interface).

Preparing a submission script

A submission script is a Linux shell script that

- describes the processing to carry out (*e.g.* the application, its input and output, etc.) and
- requests computer resources (number of GPUs, amount of memory, etc.) to use for processing.

The simplest case is that of a job that requires a single node with the following requirements:

- the job uses 1 node,
- the application is a single process,
- the job uses a single GPU,
- the job will run for no more than 10 hours,
- the job is given the name "job123" and
- the user should be emailed when the job starts and stops or aborts.

Supposing the application run is called `myCode` and takes no command line arguments, the following submission script runs the application in a single job::

```
#!/bin/bash

# set the number of nodes
#SBATCH --nodes=1

# set max wallclock time
```

```
#SBATCH --time=10:00:00

# set name of job
#SBATCH --job-name=job123

# set number of GPUs
#SBATCH --gres=gpu:1

# mail alert at start, end and abortion of execution
#SBATCH --mail-type=ALL

# send mail to this address
#SBATCH --mail-user=john.brown@gmail.com

# run the application
myCode
```

The script starts with `#!/bin/bash` (also called a shebang), which makes the submission script a Linux bash script.

The script continues with a series of lines starting with `#`, which represent bash script comments. For Slurm, the lines starting with `#SBATCH` are directives that request job scheduling resources. (Note: it is important that you put all the directives at the top of a script, before any other commands; any `#SBATCH` directive coming after a bash script command is ignored!)

The resource request `#SBATCH --nodes=n` determines how many compute nodes a job are allocated by the scheduler; only 1 node is allocated for this job.

The maximum walltime is specified by `#SBATCH --time=T`, where `T` has format `H:M:S`. Normally, a job is expected to finish before the specified maximum walltime. After the walltime reaches the maximum, the job terminates regardless whether the job processes are still running or not.

The name the job is identified by in the queue can be specified too with `#SBATCH --job-name=name`.

Lastly, an email notification is sent if an address is specified with `#SBATCH --mail-user=<email_address>`. The notification options can be set with `#SBATCH --mail-type=<type>`, where `<type>` may be `BEGIN`, `END`, `FAIL`, `REQUEUE` or `ALL` (for any change of job state).

The final part of a script is normal Linux bash script and describes the set of operations to follow as part of the job. The job starts in the same folder where it was submitted (unless an alternative path is specified), and with the same environment variables (modules, etc.) that the user had at the time of the submission. In this example, this final part only involves invoking the `myCode` application executable.

Submitting jobs with the command sbatch

Once you have a submission script ready (*e.g* called `submit.sh`), the job is submitted to the execution queue with the command `sbatch script.sh`. The queueing system prints a number (the job id) almost immediately and returns control to the linux prompt. At this point the job is in the submission queue.

Once the job submitted, it will sit in a pending state until the resources have been allocated to your job (the length of time your job is in the pending state is dependent upon a number of factors including how busy the system is and what resources you are requesting). You can monitor the progress of the job using the command `squeue` (see below).

Once the job starts to run you will see files with names such as `slurm-1234.out` either in the directory you submitted the job from (default behaviour) or in the directory where the script was instructed explicitly to change to.

Job partitions on JADE

Partitions are Slurm entities defined by the system administrators that allow the separation and control of jobs according to their characteristics. Each partition has a a number of compute nodes associated with it, as well as properties that control job placement. A job can be submitted to be executed by a particular partition, and if no partition is specified, the default one is selected.

There are three partitions on JADE, which are:

Partition name	Description
`big`	Partition dedicated to jobs that occupy an entire node, *i.e.* 8 GPUs
`small`	Partition dedicated to jobs that utilise a single GPUs each.
`devel`	Partition dedicated to testing.

The partitions have the following limits for submitted jobs:

Partition name	Partition Size	Job Walltime limit	Running Job limit
`big`	11 nodes	24 hours	5 Jobs
`small`	10 nodes	6 days	8 Jobs
`devel`	1 node	1 hour	1 Job

The default partition is `big`. Information on these partitions can be obtained with the commands `sinfo -a` or `scontrol show partition=small`.

Submitting to a particular partition can be done by specifying the partition as an argument to `sbatch`, *e.g.* `sbatch -p devel sub.sh`, or by directly supplying a request for that partition in the submission script, *e.g.* `#SBATCH --partition=devel`.

The `devel` partition should be used to check your submission script works correctly and that your application starts to execute without errors.

Upon reaching the per user running job limit for a partition, any further jobs submitted to that same partition by the same user will be shown as state Pending (PD) with the Reason set as QOSMaxJobsPerUserLimit.

Monitoring jobs with the command squeue

`squeue` is the main command for monitoring the state of systems, groups of jobs or individual jobs.

The command `squeue` prints the list of current jobs. The list looks something like this:

```
| JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
|  2497     devel     srun      bob  R       0:07      1 dgj416
|  2499       big     test1    mary  R       0:22      4 dgj[101,104]
|  2511     small     test2   steve PD       0:00      4 (Resources)
```

The first column gives the job ID, the second the partition where the job was submitted, the third the name of the job (specified by the user in the submission script) and the fourth the user ID of the job owner. The fifth is the status of the job (**R** = running, **PD** = pending, **CA** = cancelled, **CF** = configuring, **CG** = completing, **CD** = completed, **F** = failed). The sixth column gives the elapsed time for each particular job. Finally, there are the number of nodes requested and the nodelist where the job is running (or the cause that it is not running).

Some useful command line options for `squeue` include:

- `-u` for showing the status of all the jobs of a particular user, *e.g.* `squeue -u bob`;
- `-l` for showing more of the available information;
- `-j` for showing information regarding a particular job ID, *e.g.* `squeue -j 7890`;
- `--start` to report the expected start time of pending jobs.

Read all the options for squeue on the Linux manual using the command `man squeue`, including how to personalize the information to be displayed.

Deleting jobs with the command scancel

Use the `scancel` command to delete a job, *e.g.* `scancel 1121` to delete job with ID **1121**. Any user can delete their own jobs at any time, whether the job is pending (waiting in the queue) or running. A user cannot delete the jobs of another user. Normally, there is a (small) delay between the execution of the `scancel` command and the time when the job is dequeued and killed.

Environment variables

At the time a job is launched into execution, Slurm defines multiple environment variables, which can be used from within the submission script to define the correct workflow of the job. A few useful environment variables are the following:

- `SLURM_SUBMIT_DIR`, which points to the directory where the sbatch command is issued;
- `SLURM_JOB_NODELIST`, which returns the list of nodes allocated to the job;
- `SLURM_JOB_ID`, which is a unique number Slurm assigns to a job.

In most cases, `SLURM_SUBMIT_DIR` does not have to be used, as the job lands by default in the directory where the Slurm command `sbatch` was issued.

`SLURM_SUBMIT_DIR` can be useful in a submission script when files must be copied to/from a specific directory that is different from the directory where `sbatch` was issued.

`SLURM_JOB_ID` is useful to tag job specific files and directories (typically output files or run directories) in order to identify them as produced by a particular job. For instance, the submission script line

```
myApp &> $SLURM_JOB_ID.out
```

runs the application myApp and redirects the standard output (and error) to a file whose name is given by the job ID. *Note*: the job ID is a number assigned by Slurm and differs from the character string name given to the job in the submission script by the user.

Job arrays

Job arrays is a useful mechanism for submitting and managing collections of similar jobs quickly and easily; multiple job are submitted to the queue using a single `sbatch` command and a single submission script.

Here are a few examples::

```
# submit a job array with index values between 0 and 7
$ sbatch --array=0-7 sub.sh

# submit a job array with index values of 1, 3, 5 and 7
$ sbatch --array=1,3,5,7 sub.sh

# submit a job array with index values between 1 and 7 with a step size of 2 (i.e. 1,␣
↪3, 5 and 7)
$ sbatch --array=1-7:2 sub.sh
```

The index values are used by Slurm to initialise two environment variables when the job launches into execution. These variables are

- SLURM_ARRAY_JOB_ID, set to the first job ID of the array and
- SLURM_ARRAY_TASK_ID, set to the job array index value.

To give an example, suppose you submit an array of three jobs using the submission command `sbatch --array=1-3 sub.sh`, which returns:

```
Submitted batch job 10
```

Then, the environment variables in the three jobs will be

Job array index	Variables
1	SLURM_ARRAY_JOB_ID=10; SLURM_ARRAY_TASK_ID=1
2	SLURM_ARRAY_JOB_ID=10; SLURM_ARRAY_TASK_ID=2
3	SLURM_ARRAY_JOB_ID=10; SLURM_ARRAY_TASK_ID=3

The above environment variables can be used within the submission script to define what each individual job within the array does. To take a simple example, suppose each job in the array uses a single GPU and takes the input from a file that is identified by the same index as the job. The submission script could look like this:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --job-name=test
#SBATCH --time=00:30:00
#SBATCH --gres=gpu:1

myCode --input "file_${SLURM_ARRAY_TASK_ID}.inp"
```

To reiterate, the advantage of using job arrays is a single job script as the one above can be used to launch a large number of jobs, each working on a different tasks, in a controlled way.

1.1.6 Using Containerised Applications

On entering the container the present working directory will be the user's home directory: `/home_directory`

Any files you copy into `/home_directory` will have the same userid as normal and will be available once exiting the container. The local disk space on the node is available at: `/local_scratch/$USERID`

This is 6.6TB in size but any data will be lost once the interactive session is ended. There are two ways of interacting with the containerised applications.

Docker Containers

Listing available containers

To list the containers and version available on the system do:

```
root@dgj223:~# containers
REPOSITORY                    TAG                 IMAGE ID            CREATED
↪   SIZE
nvcr.io/nvidia/caffe          17.11               74f90888fb24        4 weeks ago
↪   3.247 GB
nvcr.io/nvidia/theano         17.11               39aed30f94ed        6 weeks ago
↪   3.367 GB
nvcr.io/nvidia/torch          17.11               1fb9e886f48c        6 weeks ago
↪   3.267 GB
```

```
nvcr.io/nvidia/caffe2       17.10               2ff2ccd2c8c1        10 weeks ago
↪  2.731 GB
nvcr.io/nvidia/tensorflow   17.07               94b1afe1821c        5 months ago
↪  4.404 GB
nvidia/cuda                 latest              15e5dedd88c5        7 months ago
↪  1.67 GB
nvcr.io/nvidia/caffe        17.04               87c288427f2d        7 months ago
↪  2.794 GB
nvcr.io/nvidia/theano       17.04               24943feafc9b        8 months ago
↪  2.386 GB
nvcr.io/nvidia/torch        17.04               a337ffb42c8e        8 months ago
↪  2.9 GB
Last updated:Tue Dec 19 01:00:01 GMT 2017
```

Interactive Mode

All the applications in containers can be launched interactively in the same way using 1 compute node at a time. The number of GPUs to be used per node is requested using the `gres` option. To request an interactive session on a compute node the following command is issued from the login node:

```
srun --gres=gpu:2 --pty /jmain01/apps/docker/caffe 17.04
```

This command will show the following, which is now running on a compute node:

```
================
==NVIDIA Caffe==
================

NVIDIA Release 17.04 (build 26740)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright (c) 2014, 2015, The Regents of the University of California (Regents)
All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying
↪project or file.

groups: cannot find name for group ID 1002
I have no name!@124cf0e3582e:/home_directory$
```

Note. The warnings in the last two lines can be ignored. To exit the container, issue the "exit" command. To launch the other containers the commands are:

```
srun --gres=gpu:8 --pty /jmain01/apps/docker/theano 17.04
srun --gres=gpu:4 --pty /jmain01/apps/docker/torch 17.04
```

Batch Mode

There are wrappers for launching the containers in batch mode. For example, to launch the Torch application change directory to where the launching script is, in this case called `submit-char.sh`:

```
cd /jmain01/home/atostest/char-rnn-master
```

A Slurm batch script is used to launch the code, such as:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH -J Torch
#SBATCH --gres=gpu:8
#SBATCH --time=01:00:00

/jmain01/apps/docker/torch-batch -c ./submit-char.sh
```

The output will appear in the slurm standard output file.

Each of the containerised applications has its own batch launching script:

```
/jmain01/apps/docker/torch-batch
/jmain01/apps/docker/caffe-batch
/jmain01/apps/docker/theano-batch
```

Singularity Containers

Singularity 2.4 is installed in `/jmain01/apps/singularity/2.4`. When you build your container, within your own environment, your container you must have the following directories:

```
/tmp
/local_scratch
```

These will be mounted by the local node when your container executes. The `/tmp` & `/local_scratch` directory are the local RAID disks on the DGX node and should be used for building code or temporary files.

Unlike Docker containers, the home directory the same as when you're outside the container (e.g. `/jmain01/home/your_project/your_group/your_username`). You can use `cd ~` to get to your home directory and `echo $HOME` to print out your home location.

There are 2 scripts in the `/jmain01/apps/singularity/2.4/bin` directory that you can use to launch your container using Slurm:

```
singbatch
singinteractive
```

You call them with either

```
singinteractive CONTAINER_FILE
# OR
singbatch CONTAINER_FILE SCRIPT_TO_EXECUTE
```

You should use these scripts with Slurm. So for example with an INTERACTIVE session:

```
module load singularity
srun -I --pty -t 0-10:00 --gres gpu:1 -p small singinteractive /jmain01/apps/
↪singularity/singularity-images/caffe-gpu.img
```

If you want to run in batch mode, you should call `singbatch` (using sbatch) and provide a script to execute within the container.

You MUST respect the `CUDA_VISIBLE_DEVICES` variable within the container, as you can see ALL the GPUs in the container. Some of these GPUs may be in use by other users and Slurm has allocated you a specific ones/group & will set this variable for you. If you are familiar with Docker, it only shows you the GPUs have been allocated.

Slurm will clear out `/tmp` and `/local_scratch` once you exit the container, so make sure you copy anything back to your home directory if you need it! There is an example "caffe" image provided in `/jmain01/apps/singularity/singularity-images` if you wish to contribute an image for others to use, please submit an issue to the Github Issue tracker

1.2 Software on JADE

The software initially installed on the machine is listed in the following table:

Application	Version	Note
GNU compiler suite	4.8.4	part of O/S
PGI compiler suite	17.4	
OpenMPI	1.10.2	Supplied with PGI
OpenMPI	1.10.5a1	Supplied with PGI
OpenMPI	2.1.2	Inlcudes multi-thread support
Gromacs	2016.3	Supplied by Nvidia
NAMD	2.12	

This software has been built from source and installed as modules. To list the source built applications do:

```
$ module avail
------------------------------------------------------ /jmain01/apps/modules ----------
↪--------------------
gromacs/2016.3           openmpi/1.10.2/2017      pgi/17.4(default)        pgi64/17.
↪4(default)
PrgEnv-pgi/17.4(default)      NAMD/2.12      openmpi/1.10.5a1/GNU     pgi/2017
↪     pgi64/2017
```

The following are the available applications, libraries and development tools on the JADE cluster:

1.2.1 Machine Learning

Caffe

Caffe

URL http://caffe.berkeleyvision.org/

Caffe is a Deep Learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors.

The Caffe Docker Container

Caffe is available on JADE through the use of a Docker container. For more information on JADE's use of containers, see *Using Containerised Applications*.

Using Caffe Interactively

All the contained applications are launched interactively in the same way within 1 compute node at a time. The number of GPUs to be used per node is requested using the "gres" option. To request an interactive session on a compute node the following command is issued from the login node:

```
# Requesting 2 GPUs for Caffe image version 17.04
srun --gres=gpu:2 --pty  /jmain01/apps/docker/caffe 17.04
```

This command will show the following, which is now running on a compute node:

```
==================
== NVIDIA Caffe ==
==================

NVIDIA Release 17.04 (build 26740)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright (c) 2014, 2015, The Regents of the University of California (Regents)
All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.

groups: cannot find name for group ID 1002
I have no name!@124cf0e3582e:/home_directory$
```

Note: The group ID warning and no name warning can safely be ignored.

Note: Inside the container, your home directory on the outside e.g. `/jmain01/home/JAD00X/test/test1-test` is mapped to the `/home_directory` folder inside the container.

You can test this by using the command: ls /home_directory

You are now inside the container where `Caffe` is installed. Let's check by asking for the version:

```
caffe --version
```

Where you will get something like:

```
caffe version 0.16.0
```

You can now begin training your network as normal:

```
caffe train -solver=my_solver.prototxt
```

Using Caffe in Batch Mode

There are wrappers for launching the containers within batch mode.

Firstly navigate to the folder you wish your script to lauch from, for example we'll use the home directory:

```
cd ~
```

It is recommended that you create a **script file** e.g. `script.sh`:

```
#!/bin/bash

# Prints out Caffe's version number
caffe --version
```

And don't forget to make your `script.sh` executable:

```
chmod +x script.sh
```

Then create a **Slurm batch script** that is used to launch the code, e.g. `batch.sh`:

```
#!/bin/bash

# set the number of nodes
#SBATCH --nodes=1

# set max wallclock time
#SBATCH --time=01:00:00

# set name of job
#SBATCH -J JobName

# set number of GPUs
#SBATCH --gres=gpu:8

# mail alert at start, end and abortion of execution
#SBATCH --mail-type=ALL

# send mail to this address
#SBATCH --mail-user=your.mail@yourdomain.com


#Launching the commands within script.sh
/jmain01/apps/docker/caffe-batch -c ./script.sh
```

You can then submit the job using `sbatch`:

```
sbatch batch.sh
```

On successful submission, a job ID is given:

```
Submitted batch job 7800
```

The output will appear in the slurm standard output file with the corresponding job ID (in this case `slurm-7800.out`). The content of the output is as follows:

```
==================
== NVIDIA Caffe ==
==================

NVIDIA Release 17.04 (build 26740)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright (c) 2014, 2015, The Regents of the University of California (Regents)
```

```
All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.

caffe version 0.16.0
```

Tensorflow

Tensorflow

URL https://www.tensorflow.org/

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

Tensorflow Docker Container

Tensorflow is available on JADE through the use of a Docker container. For more information on JADE's use of containers, see *Using Containerised Applications*.

Using Tensorflow Interactively

All the contained applications are launched interactively in the same way within 1 compute node at a time. The number of GPUs to be used per node is requested using the "gres" option. To request an interactive session on a compute node the following command is issued from the login node:

```
# Requesting 2 GPUs for Tensorflow image version 17.07
srun --gres=gpu:2 --pty  /jmain01/apps/docker/tensorflow 17.07
```

This command will show the following, which is now running on a compute node:

```
================
== TensorFlow ==
================

NVIDIA Release 17.07 (build 84991)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright 2017 The TensorFlow Authors.  All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.
```

```
groups: cannot find name for group ID 30773
I have no name!@d129dbb678f2:/home_directory$
```

Note: The group ID warning and no name warning can safely be ignored.

Note: Inside the container, your home directory on the outside e.g. `/jmain01/home/JAD00X/test/test1-test` is mapped to the `/home_directory` folder inside the container.

You can test this by using the command: ls /home_directory

You can test that `Tensorflow` is running on the GPU with the following python code `tftest.py`

```
import tensorflow as tf
# Creates a graph.
with tf.device('/gpu:0'):
  a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
  b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
  c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

Run the `tftest.py` script with the following command:

```
python tftest.py
```

Which gives the following results:

```
[[ 22.  28.]
 [ 49.  64.]]
```

Using Tensorflow in Batch Mode

There are wrappers for launching the containers within batch mode.

Firstly navigate to the folder you wish your script to lauch from, for example we'll use the home directory:

```
cd ~
```

It is recommended that you create a **script file** e.g. `script.sh`:

```
#!/bin/bash

# Run the tftest.py script, see previous section for contents
python tftest.py
```

And don't forget to make your `script.sh` executable:

```
chmod +x script.sh
```

Then create a **Slurm batch script** that is used to launch the code, e.g. `batch.sh`:

```
#!/bin/bash

# set the number of nodes
#SBATCH --nodes=1

# set max wallclock time
#SBATCH --time=01:00:00

# set name of job
#SBATCH -J JobName

# set number of GPUs
#SBATCH --gres=gpu:8

# mail alert at start, end and abortion of execution
#SBATCH --mail-type=ALL

# send mail to this address
#SBATCH --mail-user=your.mail@yourdomain.com


#Launching the commands within script.sh
/jmain01/apps/docker/tensorflow-batch -c ./script.sh
```

You can then submit the job using `sbatch`:

```
sbatch batch.sh
```

On successful submission, a job ID is given:

```
Submitted batch job 7800
```

The output will appear in the slurm standard output file with the corresponding job ID (in this case `slurm-7800.out`). The content of the output is as follows:

```
================
== TensorFlow ==
================

NVIDIA Release 17.07 (build 84991)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright 2017 The TensorFlow Authors.  All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.

[[ 22.  28.]
       [ 49.  64.]]
```

Using multiple GPUs

Example taken from tensorflow documentation.

If you would like to run TensorFlow on multiple GPUs, you can construct your model in a multi-tower fashion where each tower is assigned to a different GPU. For example:

```
import tensorflow as tf
# Creates a graph.
c = []
for d in ['/gpu:2', '/gpu:3']:
  with tf.device(d):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
    c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
  sum = tf.add_n(c)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print sess.run(sum)
```

You will see something similar to the following output.

```
Device mapping:
/job:localhost/replica:0/task:0/gpu:0 -> device: 0, name: Tesla K20m, pci bus
id: 0000:02:00.0
/job:localhost/replica:0/task:0/gpu:1 -> device: 1, name: Tesla K20m, pci bus
id: 0000:03:00.0
/job:localhost/replica:0/task:0/gpu:2 -> device: 2, name: Tesla K20m, pci bus
id: 0000:83:00.0
/job:localhost/replica:0/task:0/gpu:3 -> device: 3, name: Tesla K20m, pci bus
id: 0000:84:00.0
Const_3: /job:localhost/replica:0/task:0/gpu:3
Const_2: /job:localhost/replica:0/task:0/gpu:3
MatMul_1: /job:localhost/replica:0/task:0/gpu:3
Const_1: /job:localhost/replica:0/task:0/gpu:2
Const: /job:localhost/replica:0/task:0/gpu:2
MatMul: /job:localhost/replica:0/task:0/gpu:2
AddN: /job:localhost/replica:0/task:0/cpu:0
[[  44.   56.]
 [  98.  128.]]
```

Theano

Theano

URL http://deeplearning.net/software/theano/index.html

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano is most commonly used to perform Deep Learning and has excellent GPU support and integration through PyCUDA. The following steps can be used to setup and configure Theano on your own profile.

Theano Docker Container

Theano is available on JADE through the use of a Docker container. For more information on JADE's use of containers, see *Using Containerised Applications*.

Using Theano Interactively

All the contained applications are launched interactively in the same way within 1 compute node at a time. The number of GPUs to be used per node is requested using the "gres" option. To request an interactive session on a compute node the following command is issued from the login node:

```
# Requesting 2 GPUs for Theano image version 17.04
srun --gres=gpu:2 --pty  /jmain01/apps/docker/theano 17.04
```

This command will show the following, which is now running on a compute node:

```
============
== Theano ==
============

NVIDIA Release 17.04 (build 21022)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright (c) 2008--2016, Theano Development Team
All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.

groups: cannot find name for group ID 30773
I have no name!@8ded1f005768:/home_directory$
```

Note: The group ID warning and no name warning can safely be ignored.

Note: Inside the container, your home directory on the outside e.g. `/jmain01/home/JAD00X/test/test1-test` is mapped to the `/home_directory` folder inside the container.

You can test this by using the command: ls /home_directory

You can test that `Theano` is running on the GPU with the following python code `theanotest.py`

```
from theano import function, config, shared, sandbox
import theano.tensor as T
import numpy
import time

vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
iters = 1000

rng = numpy.random.RandomState(22)
x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
f = function([], T.exp(x))
print(f.maker.fgraph.toposort())
t0 = time.time()
for i in range(iters):
    r = f()
t1 = time.time()
print("Looping %d times took %f seconds" % (iters, t1 - t0))
```

```
print("Result is %s" % (r,))
if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):
    print('Used the cpu')
else:
    print('Used the gpu')
```

Run the `theanotest.py` script with the following command:

```
THEANO_FLAGS="device=gpu" python theanotest.py
```

The `THEANO_FLAGS` `device` variable can be set to either `cpu` or `gpu`.

Which gives the following results:

```
WARNING (theano.sandbox.cuda): The cuda backend is deprecated and will be removed in␣
↪the next release (v0.10).  Please switch to the gpuarray backend. You can get more␣
↪information about how to switch at this URL:
https://github.com/Theano/Theano/wiki/Converting-to-the-new-gpu-back-end%28gpuarray%29

Using gpu device 0: Tesla P100-SXM2-16GB (CNMeM is enabled with initial size: 90.0%␣
↪of memory, cuDNN 6020)
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>),␣
↪HostFromGpu(GpuElemwise{exp,no_inplace}.0)]
Looping 1000 times took 0.230759 seconds
Result is [ 1.23178029  1.61879349  1.52278066 ...,  2.20771813  2.29967761
  1.62323296]
Used the gpu
```

Note: Theano's cuda backend warning can be ignored for now.

Using Theano in Batch Mode

There are wrappers for launching the containers within batch mode.

Firstly navigate to the folder you wish your script to lauch from, for example we'll use the home directory:

```
cd ~
```

It is recommended that you create a **script file** e.g. `script.sh`:

```
#!/bin/bash

# Run the theanotest.py script, see previous section for contents
python theanotest.py
```

And don't forget to make your `script.sh` executable:

```
chmod +x script.sh
```

Then create a **Slurm batch script** that is used to launch the code, e.g. `batch.sh`:

```
#!/bin/bash

# set the number of nodes
#SBATCH --nodes=1
```

```
# set max wallclock time
#SBATCH --time=01:00:00

# set name of job
#SBATCH -J JobName

# set number of GPUs
#SBATCH --gres=gpu:8

# mail alert at start, end and abortion of execution
#SBATCH --mail-type=ALL

# send mail to this address
#SBATCH --mail-user=your.mail@yourdomain.com


#Launching the commands within script.sh
/jmain01/apps/docker/theano-batch -c ./script.sh
```

You can then submit the job using `sbatch`:

```
sbatch batch.sh
```

On successful submission, a job ID is given:

```
Submitted batch job 7800
```

The output will appear in the slurm standard output file with the corresponding job ID (in this case `slurm-7800.out`). The content of the output is as follows:

```
============
== Theano ==
============

NVIDIA Release 17.04 (build 21022)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright (c) 2008--2016, Theano Development Team
All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.

WARNING (theano.sandbox.cuda): The cuda backend is deprecated and will be removed in␣
↪the next release (v0.10).  Please switch to the gpuarray backend. You can get more␣
↪information about how to switch at this URL:
https://github.com/Theano/Theano/wiki/Converting-to-the-new-gpu-back-end%28gpuarray%29

Using gpu device 0: Tesla P100-SXM2-16GB (CNMeM is enabled with initial size: 90.0%␣
↪of memory, cuDNN 6020)
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>),␣
↪HostFromGpu(GpuElemwise{exp,no_inplace}.0)]
Looping 1000 times took 0.230759 seconds
Result is [ 1.23178029  1.61879349  1.52278066 ...,  2.20771813  2.29967761
  1.62323296]
Used the gpu
```

Torch

Torch

URL http://torch.ch/

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation.

Torch Docker Container

Torch is available on JADE through the use of a Docker container. For more information on JADE's use of containers, see *Using Containerised Applications*.

Using Torch Interactively

All the contained applications are launched interactively in the same way within 1 compute node at a time. The number of GPUs to be used per node is requested using the "gres" option. To request an interactive session on a compute node the following command is issued from the login node:

```
# Requesting 2 GPUs for Torch image version 17.04
srun --gres=gpu:2 --pty  /jmain01/apps/docker/torch 17.04
```

This command will show the following, which is now running on a compute node:

```
  _______                    __   |  Torch7
 /_  __/__  ________ _____/ /   |  Scientific computing for Lua.
  / / / _ \/ __/ __/ _ \  |
 /_/  \___/_/  \__/_//_/  |  https://github.com/torch
                          |  http://torch.ch

NVIDIA Release 17.04 (build 17724)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright (c) 2016, Soumith Chintala, Ronan Collobert, Koray Kavukcuoglu, Clement␣
↪Farabet
All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.

groups: cannot find name for group ID 30773
I have no name!@f1915084ec5f:/home_directory$
```

Note: The group ID warning and no name warning can safely be ignored.

Note: Inside the container, your home directory on the outside e.g. `/jmain01/home/JAD00X/test/test1-test` is mapped to the `/home_directory` folder inside the container.

You can test this by using the command: ls /home_directory

You are now inside the container where `Torch` is installed.

Torch console

`Torch` can be used interactively by using the `th` command:

```
th
```

Where you will the torch command prompt:

```
  _______                __   |  Torch7
 /_  __/__  ________/ /   |  Scientific computing for Lua.
  / / / _ \/ __/ __/ _ \  |  Type ? for help
 /_/  \___/_/  \__/_//_/  |  https://github.com/torch
                          |  http://torch.ch

th>
```

When you're done, type `exit` and then `y` to exit the `Torch` console:

```
th> exit
Do you really want to exit ([y]/n)? y
I have no name!@f1915084ec5f:/home_directory$
```

Using LUA script

It is also possible to pass a LUA script to the `th` command. For example, create a `test.lua` file in the current directory with the contents:

```
torch.manualSeed(1234)
-- choose a dimension
N = 5

-- create a random NxN matrix
A = torch.rand(N, N)

-- make it symmetric positive
A = A*A:t()

-- make it definite
A:add(0.001, torch.eye(N))

-- add a linear term
b = torch.rand(N)

-- create the quadratic form
function J(x)
   return 0.5*x:dot(A*x)-b:dot(x)
end

print(J(torch.rand(N)))
```

Call the `test.lua` script by using the command:

```
th test.lua
```

Which shows the following results:

```
0.72191523289161
```

Using Torch in Batch Mode

There are wrappers for launching the containers within batch mode.

Firstly navigate to the folder you wish your script to lauch from, for example we'll use the home directory:

```
cd ~
```

It is recommended that you create a **script file** e.g. `script.sh`:

```
#!/bin/bash

# Runs a script called test.lua
# see above section for contents
th test.lua
```

And don't forget to make your `script.sh` executable:

```
chmod +x script.sh
```

Then create a **Slurm batch script** that is used to launch the code, e.g. `batch.sh`:

```
#!/bin/bash

# set the number of nodes
#SBATCH --nodes=1

# set max wallclock time
#SBATCH --time=01:00:00

# set name of job
#SBATCH -J JobName

# set number of GPUs
#SBATCH --gres=gpu:8

# mail alert at start, end and abortion of execution
#SBATCH --mail-type=ALL

# send mail to this address
#SBATCH --mail-user=your.mail@yourdomain.com

#Launching the commands within script.sh
/jmain01/apps/docker/torch-batch -c ./script.sh
```

You can then submit the job using `sbatch`:

```
sbatch batch.sh
```

On successful submission, a job ID is given:

```
Submitted batch job 7800
```

The output will appear in the slurm standard output file with the corresponding job ID (in this case `slurm-7800.out`). The content of the output is as follows:

```
  _______                     __      |  Torch7
 /_  __/__  ________/ /      |  Scientific computing for Lua.
  / / / _ \/ __/ __/ _ \     |
 /_/  \___/_/  \__/_//_/     |  https://github.com/torch
                             |  http://torch.ch

NVIDIA Release 17.04 (build 17724)

Container image Copyright (c) 2017, NVIDIA CORPORATION.  All rights reserved.
Copyright (c) 2016, Soumith Chintala, Ronan Collobert, Koray Kavukcuoglu, Clement␣
↪Farabet
All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying␣
↪project or file.

0.72191523289161
```

1.2.2 Molecular Dynamics

Gromacs

Gromacs

URL http://www.gromacs.org/

URL https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/gromacs/

Gromacs is a versatile package for molecular dynamics simulations, which solves the Newtonian equations of motion for systems with hundreds to millions of particles. Although the software scales well to hundreds of cores for typical simulations, Gromacs calculations are restricted to at most a single node on the JADE service.

Job scripts

Gromacs jobs can run using 1, 4 (half a node), or 8 (full node) GPUs (please see note below regarding job performance). The following Slurm script example is written for one of the regression tests from the installation:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=5
#SBATCH --gres=gpu:1
#SBATCH --gres-flags=enforce-binding
#SBATCH --time=10:00:00
```

```
#SBATCH -J testGromacs
#SBATCH -p small

module purge
module load gromacs/2018.0

mpirun -np ${SLURM_NTASKS_PER_NODE} --bind-to socket \
       mdrun_mpi -s topol.tpr \
       -ntomp ${SLURM_CPUS_PER_TASK} &> run-gromacs.out
```

The example utilises 1 GPU (–gres=gpu:1) on a JADE node. Gromacs is started with a number of MPI processes (–ntasks-per-node=1) , which must match the number of requested GPUs. Each MPI process will run 5 OMP threads (–cpus-per-task=5). The number of requested MPI processes is saved in the environment variable SLURM_NTASKS_PER_NODE, while the number of threads per process is saved in SLURM_CPUS_PER_TASK.

The request –bind-to socket is specific to OpenMPI, which was used to build Gromacs on JADE. This extra option to the OpenMPI mpirun is essential in obtaining the optimal run configuration and computational performance.

To run the same job on 4 or 8 GPUs, you need to change the values of –ntasks-per-node and –gres=gpu from 1 to 4 or 8, respectively. You also need to change the partition from small to big. While running on 4 or 8 GPUs might increase the performance of a single job, **please note** that in terms of aggregate simulation time, it is more efficient to run single-GPU jobs. For example, a 4-GPU simulation of a 240,000 atom system yields about 18.5 ns/day using Gromacs2016.3, while a single-GPU simulation yields 8.4 ns/day. Thus, running four single-GPU simulations will provide almost double the throughput (4x8.4 ns/day = 33.6 ns/day) compared to a single 4-GPU simulation. More importantly, the single-GPU performance has been vastly improved in Gromacs2018, and is about double that of Gromacs2016.3. Thus, a single-GPU simulation using Gromacs2018 gives about the same performance as a 4-GPU simulation, and it is therefore **strongly advised to only run single-GPU simulations with Gromacs2018.**

To read more about Gromacs processing on GPUs, please visit https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/gromacs/ .

Caution: Run performance can be negatively affected by a deviation from the above "recipe". Please modify *#SBATCH* parameters or *mpirun* command line options only if you are sure this is necessary. A job that runs sub-optimally on half a node can affect the performance of another job on the other half of the same node, which would normally run at optimal performance on a "quiet" system.

Installation notes

The latest version of the source was used. The following build instructions were followed: http://www.nvidia.com/object/gromacs-installation.html

The code was compiled using OpenMPI v1.10.5a1 and GCC v4.8.4 with the following command:

```
CC=mpicc CXX=mpicxx cmake /jmain01/home/atostest/Building/gromacs-2016.3
-DGMX_OPENMP=ON
-DGMX_GPU=ON
-DGPU_DEPLOYMENT_KIT_ROOT_DIR=/usr/local/cuda-8.0/targets/x86_64-linux
-DCUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-8.0/targets/x86_64-linux
-DNVML_INCLUDE_DIR=/usr/local/cuda-8.0/targets/x86_64-linux/include
-DNVML_LIBRARY=/usr/lib/nvidia-375/libnvidia-ml.so
-DHWLOC_INCLUDE_DIRS=/usr/mpi/gcc/openmpi-1.10.5a1/include/openmpi/opal/mca/hwloc/
↪hwloc191/hwloc/include
-DGMX_BUILD_OWN_FFTW=ON
-DGMX_PREFER_STATIC_LIBS=ON
-DCMAKE_BUILD_TYPE=Release
-DGMX_BUILD_UNITTESTS=ON
-DCMAKE_INSTALL_PREFIX=/jmain01/home/atostest/gromacs-2016.3
```

NAMD

NAMD

URL http://www.ks.uiuc.edu/Research/namd/

URL https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/namd/

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. NAMD scales to hundreds of cores for typical simulations, however NAMD calculations are restricted to at most a single node on the JADE service.

Job scripts

Below is an example of a NAMD job script

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntassk-per-node=20
#SBATCH -J testNAMD
#SBATCH --time=01:00:00
#SBATCH --gres=gpu:4

module load NAMD/2.12

$NAMDROOT/namd2 +p$SLURM_NTASKS_PER_NODE +setcpuaffinity +devices $CUDA_VISIBLE_
↪DEVICES ./input.conf &> run.log
```

The above example utilises half the resources on a JADE node, with a requests for a single node with 20 tasks and 4 GPUs.

Because the job is run on a single node, NAMD can be started directly, thus avoiding the use of the launcher *charmrun*. The application is set to run on the resources allocated uwing the *+p* and *+devices* command line options. Additionally, affinity is requested using the option *+setcpuaffinity*.

The general recommendation is to have no more than one process per GPU in the multi-node run, allowing the full utilisation of multiple cores via multi-threading. For single node jobs, the use of multiple GPUs per process is permitted.

To read more about NAMD processing on GPUs, please visit https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/namd/ .

Installation notes

The latest version of the source code was used and built using OpenMPI v1.10.5a1 and GCC v4.8.4 following instructions from http://www.nvidia.com/object/gpu-accelerated-applications-namd-installation.html

Charm++ was built using:

```
./build charm++ verbs-linux-x86_64 gcc smp --with-production
```

NAMD was built using the following:

```
./config Linux-x86_64-g++ --charm-arch verbs-linux-x86_64-smp-gcc --with-cuda --cuda-
↪prefix /usr/local/cuda-8.0
```

For the decision on the number of threads to use per node, take a look at http://www.nvidia.com/object/gpu-accelerated-applications-namd-running-jobs.html

1.2.3 Python

Python

URL https://python.org

URL https://www.anaconda.com

This page documents the python and Anaconda installation on ShARC. This is the recommended way of using Python, and the best way to be able to configure custom sets of packages for your use.

"conda" a Python package manager, allows you to create "environments" which are sets of packages that you can modify. It does this by installing them in your home area. This page will guide you through loading conda and then creating and modifying environments so you can install and use whatever Python packages you need.

Using standard Python

Standard Python 2 and 3 are available to be loaded as a module:

```
python2/2.7.14
python3/3.6.3
```

Use the `module load` command to load a particular version of python e.g. for Python 2.7.14:

```
module load python2/2.7.14
```

Using conda Python

Conda version `4.3.30` is available for both Python 2 and 3 and can be loaded through provided module files:

```
apps/python2/anaconda
apps/python3/anaconda
```

Use the `module load` command to load a particular Anaconda Python version e.g. Anaconda for Python 3:

```
module load apps/python3/anaconda
```

Using conda Environments

There are a small number of environments provided for everyone to use, these are the default `root` and `python2` environments as well as various versions of Anaconda for Python 3 and Python 2.

Once the conda module is loaded you have to load or create the desired conda environments. For the documentation on conda environments see the conda documentation.

You can load a conda environment with:

```
source activate python2
```

where `python2` is the name of the environment, and unload one with:

```
source deactivate
```

which will return you to the `root` environment.

It is possible to list all the available environments with:

```
conda env list
```

Provided system-wide are a set of anaconda environments, these will be installed with the anaconda version number in the environment name, and never modified. They will therefore provide a static base for derivative environments or for using directly.

Creating an Environment

Every user can create their own environments, and packages shared with the system-wide environments will not be reinstalled or copied to your file store, they will be `symlinked`, this reduces the space you need in your `/home` directory to install many different Python environments.

To create a clean environment with just Python 2 and numpy you can run:

```
conda create -n mynumpy python=2.7 numpy
```

This will download the latest release of Python 2.7 and numpy, and create an environment named `mynumpy`.

Any version of Python or list of packages can be provided:

```
conda create -n myscience python=3.5 numpy=1.8.1 scipy
```

If you wish to modify an existing environment, such as one of the anaconda installations, you can `clone` that environment:

```
conda create --clone anaconda3-4.2.0 -n myexperiment
```

This will create an environment called `myexperiment` which has all the anaconda 4.2.0 packages installed with Python 3.

Installing Packages Inside an Environment

Once you have created your own environment you can install additional packages or different versions of packages into it. There are two methods for doing this, `conda` and `pip`, if a package is available through conda it is strongly recommended that you use conda to install packages. You can search for packages using conda:

```
conda search pandas
```

then install the package using:

```
conda install pandas
```

if you are not in your environment you will get a permission denied error when trying to install packages, if this happens, create or activate an environment you own.

If a package is not available through conda you can search for and install it using pip, *i.e.*:

```
pip search colormath

pip install colormath
```

1.3 CUDA

CUDA

URL http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html

CUDA is a parallel computing platform and API model created and developed by Nvidia, which enables dramatic increases in computing performance by harnessing the power of GPUs

1.3.1 Versions

Multiple CUDA versions are available through the module system. Version 9.0 is the current usable version following upgrade to DGX-1 Server 3.1.4. Upgrades to CUDA will be released as they become Generally Available.

1.3.2 Environment

The CUDA environment is managed through the modules, which set all the environment variables needed. The availability of different versions can be checked with

```
module avail cuda
```

The environment set by a particular module can be inspected, *e.g.*

```
module show cuda/8.0
```

1.3.3 Learn more

To learn more about CUDA programming, either talk to your local RSE support, or visit Mike Giles' CUDA Programming course page at

http://people.maths.ox.ac.uk/gilesm/cuda/

This one-week course is taught in Oxford at the end of July each year, but all of the lecture notes and practicals are provided online for self-study at other times.

1.3.4 Official CUDA documentation

NVIDIA provides lots of documentation, both online and in downloadable form:

- Online CUDA documentation

- CUDA homepage
- CUDA Runtime API
- CUDA C Best Practices Guide
- CUDA Compiler Driver NVCC
- CUDA Visual Profiler
- CUDA-gdb debugger
- CUDA-memcheck memory checker
- CUDA maths library
- CUBLAS library
- CUFFT library
- CUSPARSE library
- CURAND library
- NCCL multi-GPU communications library
- NVIDIA blog article
- GTC 2015 presentation on NCCL
- PTX (low-level instructions)

Nsight is NVIDIA's integrated development environment:

- Nsight Visual Studio
- Nsight Eclipse
- Nsight Eclipse – Getting Started

NVIDIA also provide helpful guides on the Pascal architecture:

- Pascal Tuning Guide
- Pascal P100 White Paper

Useful presentations at NVIDIA's 2017 GTC conference contain:

- Cooperative Groups
- NCCL 2.0
- Multi-GPU Programming
- The Making of Saturn-V

1.4 More Information

JADE Web site: http://www.arc.ox.ac.uk/content/jade

Mike Giles' Web site: http://people.maths.ox.ac.uk/~gilesm/JADE/

STFC Web site: https://www.hartree.stfc.ac.uk/Pages/Hartree-Centre-welcomes-new-HPC-computing-facility-to-support-machine-lear
aspx

1.5 Troubleshooting

1.5.1 Help

Univesities provide coordinated institutional technical support to assist local users and their developers with GPU applications and in accessing the JADE HPC facility. For more information, contact Research Software Engineers at your home univestrity.

1.5.2 Documentation

The source is available on GitHub and contributions are always encouraged. Contributions can be as simple as minor tweaks to this documentation, the website or the core. To contribute, fork the project on GitHub and send a pull request.

1.5.3 JADE status

The operational status of the JADE system can be checked on the main Hartree Centre status page.