GPUHackShef Documentation

Release

Mozhgan K. Chimeh

Jul 22, 2020

Contents

The University Of Sheffield.

NVIDIA®

OpenACC

More Science, Less Programming

This is the documentation for the Sheffield Virtual GPU Hackathon 2020.

You may find it useful to read the helpful GPU Hackathon Attendee Guide.

The site encourages user contributions via GitHub and has useful information for how to access the GPU systems used for the Hackathon event.

CHAPTER 1

The NVIDIA Hackathon Facility

NVIDIA have a GPU Hackathon cluster which will be used to support the hackathon.

The documentation for access is available below

- Version 1.0
- Version 2.0 (with virtual desktop)

CHAPTER 2

The Bede Facility

Bede is a new EPSRC-funded Tier 2 (regional) HPC cluster. It is currently being configured and tested and one of its first uses will be to support the hackathon.

This system is particularly well suited to supporting:

- Jobs that require multiple GPUs and possibly multiple nodes
- Jobs that require much movement of data between CPU and GPU memory

NB the system was previously known as NICE-19.

2.1 Bede Hardware

- **32x Main GPU nodes, each node (IBM AC922) has:**
  - 512GB DDR4 RAM
  - 2x IBM POWER9 CPUs (and two NUMA nodes), with
  - 4x NVIDIA V100 GPUs (2 per CPU)
  - Each CPU is connected to its two GPUs via high-bandwidth, low-latency NVLink interconnects (helps if you need to move lots of data to/from GPU memory)
- **4x Inferencing Nodes:**
  - Equipped with T4 GPUs for inference tasks.
- 2x Visualisation nodes
- 2x Login nodes
- 100 Gb EDR Infiniband (high bandwith and low latency to support multi-node jobs)
- 2PB Lustre parallel file system (available over Infiniband and Ethernet network interfaces)

2.2 Getting an Account

All participants will receive an e-mail with access details. If you did not please let one of the organisers know.

2.3 Connecting to the Bede HPC system

2.3.1 Connecting to Bede with SSH

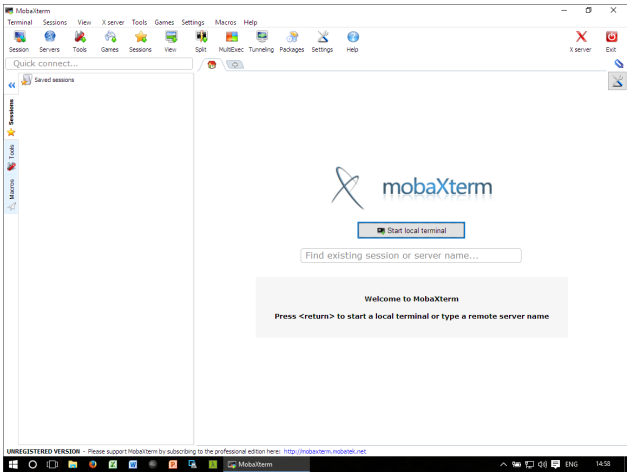
The most versatile way to **run commands and submit jobs** on one of the clusters is to use a mechanism called SSH, which is a common way of remotely logging in to computers running the Linux operating system.

To connect to another machine using SSH 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*.

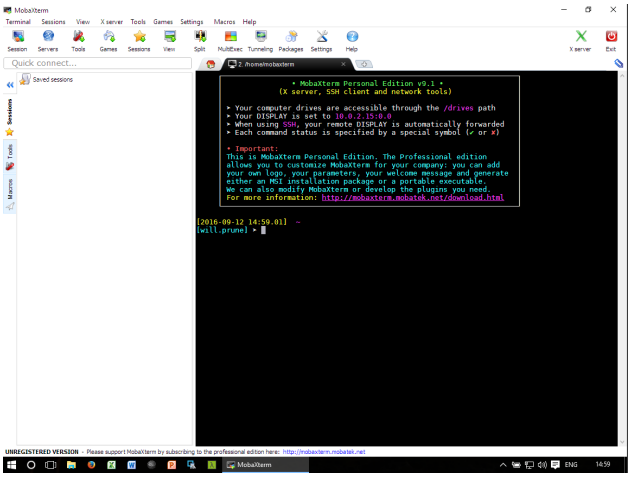
SSH client software on Windows

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 ssh.



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.

Establishing a SSH connection

Once you have a terminal open run the following command to log in to a cluster:

```
ssh $USER@login1.bede.dur.ac.uk

# Alternatively you can use the login node 2
ssh $USER@login2.bede.dur.ac.uk
```

Here you need to:

- replace `$USER` with your username (e.g. `te1st`)

You will then be asked for to enter your password. If the password is correct you should get a prompt resembling the one below:

```
(base) [te1st@login1 ~]$
```

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.

2.3.2 Transferring files

Transferring files with MobaXTerm (Windows)

After connecting to Bede with MobaXTerm, you will see a files panel of the left of the screen. You can drag files from Windows explorer into the panel to upload the file or right clicking on the files in the panel and select `Download` to download the files to your machine.

Transferring files to/from Bede with SCP (Linux and Mac OS)

Secure copy (scp) can be used to transfer files between systems through the SSH protocol.

To transfer from your machine to Bede (assuming our username is te1st):

```
# Copy myfile.txt from the current directry to your Bede home directory
scp myfile.txt te1st@login1.bede.dur.ac.uk:~/
```

To transfer from Bede to your machine:

```
# Copy myfile.txt from the Bede home directory to current local directory
scp te1st@login1.bede.dur.ac.uk:~/myfile.txt ./
```

2.4 Filestores

- `/home` - 4.9TB shared (Lustre) drive for all users.
- `/nobackup` - 2PB shared (Lustre) drive for all users.
- `/tmp` - Temporary local node SSD storage.

2.5 Running and Scheduling Tasks on Bede

2.5.1 Slurm Workload Manager

Slurm is a highly scalable cluster management and job scheduling system, used in Bede. As a cluster workload manager, Slurm has three key functions:

- it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work,
- it provides a framework for starting, executing, and monitoring work on the set of allocated nodes,
- it arbitrates contention for resources by managing a queue of pending work.

2.5.2 Loading Slurm

Slurm must first be loaded before tasks can be scheduled.

```
module load slurm
```

For more information on modules, see *Activating software using Environment Modules*.

2.5.3 Request an Interactive Shell

Launch an interactive session on a worker node using the command:

```
srun --pty bash
```

You can request an interactive node with GPU(s) by using the command:

```
# --gpus=1 requests 1 GPU for the session session, the number can be 1, 2 or 4
srun --gpus=1 --pty bash
```

You can add additional options, e.g. request additional memory:

```
# Requests 16GB of RAM with 1 GPU
srun --mem=16G --gpus=1 --pty bash
```

2.5.4 Submitting Non-Interactive Jobs

Write a job-submission shell script

You can submit your job, using a shell script. A general job-submission shell script contains the "bang-line" in the first row.

```
#!/bin/bash
```

Next you may specify some additional options, such as memory,CPU or time limit.

```
#SBATCH --"OPTION"="VALUE"
```

Load the appropriate modules if necessery.

```
module load PATH
module load MODULE_NAME
```

Finally, run your program by using the Slurm "srun" command.

```
srun PROGRAM
```

The next example script requests 2 GPUs and 16Gb memory. Notifications will be sent to an email address:

```
#!/bin/bash
#SBATCH --gpus=2
#SBATCH --mem=16G
#SBATCH --mail-user=username@mydomain.com

module load cuda

# Replace my_program with the name of the program you want to run
srun my_program
```

Job Submission

Save the shell script (let's say "submission.slurm") and use the command

```
sbatch submission.slurm
```

Note the job submission number. For example:

```
Submitted batch job 1226
```

Check your output file when the job is finished.

```
# The JOB_NAME value defaults to "slurm"
cat JOB_NAME-1226.out
```

2.5.5 Common job options

Optional parameters can be added to both interactive and non-interactive jobs. Options can be appended to the command line or added to the job submission scripts.

- **Setting maximum execution time**
  - `--time=hh:mm:ss` - Specify the total maximum execution time for the job. The default is 48 hours (48:00:00)
- **Memory request**
  - `--mem=#`- Request memory (default 4GB), suffixes can be added to signify Megabytes (M) or Gagabytes (G) e.g. `--mem=16G` to request 16GB.
  - Alternatively `--mem-per-cpu=#` or `--mem-per-gpu=#` - Memory can be requested per CPU with `--mem-per-cpu` or per GPU `--mem-per-gpu`, these three options are mutually exclusive.
- **GPU request**
  - `--gpus=1` - Request GPU(s), the number can be 1, 2 or 4.
- **CPU request**
  - `-c 1` or `--cpus-per-task=1` - Requests a number of CPUs for this job, 1 CPU in this case.
  - `--cpus-per-gpu=2` - Requests a number of CPUs **per** GPU requested. In this case we've requested 2 CPUs per GPU so if `--gpus=2` then 4 CPUs will be requested.
- **Specify output filename**
  - `--output=output.%j.test.out`
- **E-mail notification**
  - `--mail-user=username@sheffield.ac.uk` - Send notification to the following e-mail
  - `--mail-type=type` - Send notification when type is `BEGIN`, `END`, `FAIL`, `REQUEUE`, or `ALL`
- **Naming a job**
  - `--job-name="my_job_name"` - The specified name will be appended to your output (`.out`) file name.
- **Add comments to a job**
  - `--comment="My comments"`

For the full list of the available options please visit the Slurm manual webpage at https://slurm.schedmd.com/pdfs/summary.pdf.

2.5.6 Key SLURM Scheduler Commands

Display the job queue. Jobs typically pass through several states in the course of their execution. The typical states are PENDING, RUNNING, SUSPENDED, COMPLETING, and COMPLETED.

```
squeue
```

Shows job details:

```
sacct -v
```

Details the HPC nodes:

```
sinfo
```

Deletes job from queue:

```
scancel JOB_ID
```

2.6 Activating software using Environment Modules

2.6.1 Overview and rationale

'Environment Modules' are the mechanism by which much of the software is made available to the users of Bede.

To make a particular piece of software available a user will *load* a module e.g. you can load a particular version of the 'CUDA' library with:

```
module load cuda/10.2
```

This command manipulates environment variables to make this piece of software available. If you then want to switch to using a different version of CUDA (should another be installed on the cluster you are using) then you can run:

```
module unload cuda/10.2
```

then load the other.

You may wonder why modules are necessary: why not just install packages provided by the vender of the operating system installed on the cluster? In shared high-performance computing environments such as Bede:

- users typically want control over the version of applications that is used (e.g. to give greater confidence that results of numerical simulations can be reproduced);
- users may want to use applications built using compiler X rather than compiler Y as compiler X might generate faster code and/or more accurate numerical results in certain situations;
- users may want a version of an application built with support for particular parallelisation mechanisms such as MPI for distributing work between machines, OpenMP for distributing work between CPU cores or CUDA for parallelisation on GPUs);
- users may want an application built with support for a particular library.

There is therefore a need to maintain multiple versions of the same applications on Bede. Module files allow users to select and use the versions they need for their research.

If you switch to using a cluster other than Bede then you will likely find that environment modules are used there too. Modules are not the only way of managing software on clusters: increasingly common approaches include:

- the *Conda* package manager (Python-centric but can manage software written in any language);

2.6.2 Basic guide

You can list all (loaded and unloaded) modules on Bede using:

```
module avail
```

You can then load a module using e.g.:

```
module load cuda/10.2
```

You can then load further modules e.g.:

```
module load gcc/openmpi-3.0.3
```

Confirm which modules you have loaded using:

```
module list
```

If you want to stop using a module (by undoing the changes that loading that module made to your environment):

```
module unload cuda/10.2
```

or to unload all loaded modules:

```
module purge
```

You can search for a module using:

```
module avail |& grep -i somename
```

Some other things to be aware of:

- You can load and unload modules in both interactive and batch jobs;
- Modules may themselves load other modules. If this is the case for a given module then it is typically noted in our documentation for the corresponding software;
- The order in which you load modules may be significant (e.g. if module A sets `SOME_ENV_VAR=apple` and module B sets `SOME_ENV_VAR=pear`);
- Some related module files have been set up so that they are mutually exclusive e.g. on Bede the modules `cuda/10.2` and `cuda/10.1` cannot be loaded simultaneously (as users should never want to have both loaded).

2.6.3 Module Command Reference

Here is a list of the most useful `module` commands. For full details, type `man module` at the command prompt on one of the clusters.

- `module list` – lists currently loaded modules
- `module avail` – lists all available modules
- `module load modulename` – loads module `modulename`
- `module unload modulename` – unloads module `modulename`
- `module switch oldmodulename newmodulename` – switches between two modules
- `module show modulename` - Shows how loading `modulename` will affect your environment
- `module purge` – unload all modules
- `module help modulename` – may show longer description of the module if present in the modulefile
- `man module` – detailed explanation of the above commands and others

More information on the Environment Modules software can be found on the project's site.

2.7 Software on Bede

2.7.1 CUDA

CUDA (*Compute Unified Device Architecture*) is a parallel computing platform and application programming interface (API) model created by NVIDIA. It allows software developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing, an approach known as *General Purpose GPU* (GPGPU) computing.

Usage

You need to first request one or more GPUs within an *interactive session or batch job on a worker node*.

You then need to ensure a version of the CUDA library (and compiler) is loaded. CUDA version 10.1, 10.2 is currently available on Bede:

```
module load cuda/10.1
module load cuda/10.2
module load nvidia/20.5
```

The `nvidia/20.5` module contains CUDA 10.2 and additional profilers such as `ncu`.

Confirm which version of CUDA you are using via `nvcc --version` e.g.:

```
$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Thu_Oct_24_17:58:26_PDT_2019
Cuda compilation tools, release 10.2, V10.2.89
```

Compiling a simple CUDA program

An example of the use of `nvcc` (the CUDA compiler):

```
nvcc filename.cu
```

will compile the CUDA program contained in the file `filename.cu`.

Compiling the sample programs

You do not need to be using a GPU-enabled node to compile the sample programs but you do need at least one GPU to run them.

In this demonstration, we create a batch job that

1. Requests two GPUs, a single CPU core and 8GB RAM
2. Loads a module to provide CUDA 10.2
3. Downloads compatible NVIDIA CUDA sample programs
4. Compiles and runs an example that performs a matrix multiplication

```
#!/bin/bash
#SBATCH --gpus=2      # Number of GPUs
#SBATCH --mem=8G
#SBATCH --time=0-00:05        # time (DD-HH:MM)
#SBATCH --job-name=gputest

module load cuda/10.2  # provides CUDA 10.2

mkdir -p $HOME/examples
cd $HOME/examples
if ! [[ -f cuda-samples/.git ]]; then
    git clone https://github.com/NVIDIA/cuda-samples.git cuda-samples
fi
cd cuda-samples
git checkout tags/v10.2 # use sample programs compatible with CUDA 10.2
cd Samples/matrixMul
make
./matrixMul
```

GPU Code Generation Options

To achieve the best possible performance whilst being portable, GPU code should be generated for the architecture(s) it will be executed upon.

This is controlled by specifying `-gencode` arguments to NVCC which, unlike the `-arch` and `-code` arguments, allows for 'fatbinary' executables that are optimised for multiple device architectures.

Each `-gencode` argument requires two values, the *virtual architecture* and *real architecture*, for use in NVCC's two-stage compilation. I.e. `-gencode=arch=compute_70,code=sm_70` specifies a virtual architecture of `compute_70` and real architecture `sm_70`.

To support future hardware of higher compute capability, an additional `-gencode` argument can be used to enable Just in Time (JIT) compilation of embedded intermediate PTX code. This argument should use the highest virtual architecture specified in other gencode arguments for both the `arch` and `code` i.e. `-gencode=arch=compute_70,code=compute_70`.

The minimum specified virtual architecture must be less than or equal to the Compute Capability of the GPU used to execute the code.

GPU nodes in Bede contain Tesla V100 GPUs, which are compute capability 70. To build a CUDA application which targets just the public GPUS nodes, use the following `-gencode` arguments:

```
nvcc filename.cu \
   -gencode=arch=compute_70,code=sm_70 \
   -gencode=arch=compute_70,code=compute_70 \
```

Further details of these compiler flags can be found in the NVCC Documentation, along with details of the supported virtual architectures and real architectures.

Documentation

- CUDA Toolkit Documentation

Determining the NVIDIA Driver version

Run the command:

```
cat /proc/driver/nvidia/version
```

Example output is:

```
NVRM version: NVIDIA UNIX ppc64le Kernel Module  440.64.00  Wed Feb 26 16:01:28 UTC␣
↪2020
GCC version:  gcc version 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
```

2.7.2 MPI

Message Passing Interface (MPI) is a standardized and portable message-passing standard designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures.

Bede provides four MPI libraries for use depending on the application, OpenMPI, IBM Spectrum MPI, MVAPICH 2, and Mellanox HPC-X.

OpenMPI

The OpenMPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners. OpenMPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. OpenMPI offers advantages for system and software vendors, application developers and computer science researchers.

Versions

You can load a specific version using one of the following:

```
module load gcc/openmpi-3.0.3
module load gcc/openmpi-4.0.3rc4
```

IBM Spectrum MPI

IBM® Spectrum MPI is a high-performance, production-quality implementation of Message Passing Interface (MPI). It accelerates application performance in distributed computing environments. It provides a familiar portable interface based on the open-source MPI. It goes beyond Open MPI and adds some unique features of its own, such as advanced CPU affinity features, dynamic selection of interface libraries, superior workload manager integrations and better performance. IBM Spectrum MPI supports a broad range of industry-standard platforms, interconnects and operating systems, helping to ensure that parallel applications can run almost anywhere.

Versions

You can load the library using:

```
module load ibm/spectrum-mpi-10.03.01.00
```

MVAPICH2

MVAPICH2 (pronounced as "em-vah-pich 2") is an open-source MPI software to exploit the novel features and mechanisms of high-performance networking technologies (InfiniBand, 10GigE/iWARP and 10/40GigE RDMA over Converged Enhanced Ethernet (RoCE)) and deliver best performance and scalability to MPI applications. This software is developed in the Network-Based Computing Laboratory (NBCL), headed by Prof. Dhabaleswar K. (DK) Panda since 2001.

Versions

Versions are available for the GCC and IBM XL compiler:

```
module load mvapich2/gdr/2.3.4-1.mofed4.7.gnu4.8.5-2
module load mvapich2/gdr/2.3.4-1.mofed4.7.xlc16.01
```

Mellanox HPCX OpenMPI

To meet the needs of scientific research and engineering simulations, supercomputers are growing at an unrelenting rate. The Mellanox HPC-X ScalableHPC Toolkit is a comprehensive MPI and SHMEM/PGAS software suite for high performance computing environments. HPC-X provides enhancements to significantly increase the scalability and performance of message communications in the network. HPC-X enables you to rapidly deploy and deliver maximum application performance without the complexity and costs of licensed third-party tools and libraries.

Versions

You can load the library using one of the following:

```
module load gcc/hpcx/2.6.0-OFED-4.7.1.0.1/hpcx-mt-ompi
module load gcc/hpcx/2.6.0-OFED-4.7.1.0.1/hpcx-ompi
```

Using MPI

The following examples applies to all MPI libraries, change the `module load` to load your required libary.

Example

Consider the following source code (hello.c):

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
```

```
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
            processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

MPI_COMM_WORLD (which is constructed for us by MPI) encloses all of the processes in the job, so this call should return the amount of processes that were requested for the job.

Compile your source code by using one of the following commands:

```
mpic++ hello.cpp -o file
mpicxx hello.cpp -o file
mpicc hello.c -o file
mpiCC hello.c -o file
```

Interactive job submission

You can run your job interactively:

```
srun file
```

Your output would be something like:

```
Hello world from processor gpu001.bede.dur.ac.uk, rank 0 out of 1 processors
```

This is an expected behaviour since we did not specify the number of CPU cores when requesting our interactive session. You can request an interactive node with multiple cores (4 in this example) by using the command:

```
srun --ntasks=4 --pty bash -i
```

Please note that requesting multiple cores in an interactive node depends on the availability. During peak times, it is unlikely that you can successfully request a large number of cpu cores interactively. Therefore, it may be a better approach to submit your job non-interactively.

Non-interactive job submission

Write a shell script (minimal example) We name the script as 'test.sh':

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40

module load OpenMPI/3.1.3-GCC-8.2.0-2.31.1
```

```
srun --export=ALL file
```

Maximum 40 cores can be requested.

Submit your script by using the command:

```
sbatch test.sh
```

Your output would be something like:

```
Hello world from processor gpu004.bede.dur.ac.uk, rank 24 out of 40 processors
Hello world from processor gpu001.bede.dur.ac.uk, rank 5 out of 40 processors
...
Hello world from processor gpu006.bede.dur.ac.uk, rank 31 out of 40 processors
Hello world from processor gpu039.bede.dur.ac.uk, rank 32 out of 40 processors
```

2.7.3 Python and Anaconda

This page documents the "Anaconda" installation on Bede. 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 conda Python

After connecting to Bede, start an interactive session with the `srun --pty bash` command.

Anaconda Python can be loaded with:

```
module load Anaconda3/2020.02
```

The `root` conda environment (the default) provides Python 3 and no extra modules, it is automatically updated, and not recommended for general use, just as a base for your own environments.

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 3.7 and numpy you can run:

```
conda create -n mynumpy python=3.7 numpy
```

This will download the latest release of Python 3.7 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 myscience -n myexperiment
```

This will create an environment called `myexperiment` which has all the same conda packages as the `myscience` environment.

Using conda Environments

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 myexperiment
```

where `myexperiment` 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.

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
```

Using conda and Python in a batch job

Create a batch job submission script called `myscript.slurm` that is similar to the following:

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

export SLURM_EXPORT_ENV=ALL
module load Anaconda3/2019.07

# We assume that the conda environment 'myexperiment' has already been created
source activate myexperiment
srun python mywork.py
```

Then submit this to Slurm by running:

```
sbatch myscript.slurm
```

2.7.4 PyTorch

PyTorch

URL https://pytorch.org

PyTorch is an open source machine learning library for Python, based on Torch. It is used for applications such as natural language processing.

About PyTorch on Bede

Note: GPU must be requested in order to enable GPU acceleration by adding the flag e.g. `--gpus=1` to the scheduler command or job script. See *Running and Scheduling Tasks on Bede* for more information.

Bede has a locally installed IBM Watson Machine Learning Community Edition (WML CE) Anaconda channel that provides versions of Tensorflow, PyTorch and their dependencies especially built for the POWER architecture.

Installation in Home Directory

First request an interactive session, e.g. see *Running and Scheduling Tasks on Bede*.

Then PyTorch can be installed by the following

```
# Load the conda module
module load Anaconda3/2020.02

# Add the local WML CE channel to the conda search path
 conda config --prepend channels file:///opt/software/apps/ibm_wmlce/wmlce-1.7.0-
↪mirror/

# Create an conda virtual environment called e.g. named 'pytorch'
conda create -n pytorch python=3.6

# Activate the 'pytorch' environment
```

```
source activate pytorch

# Install PyTorch
conda install torch
```

Every Session Afterwards and in Your Job Scripts

Every time you use a new session or within your job scripts, the modules must be loaded and conda must be activated again. Use the following command to activate the Conda environment with PyTorch installed:

```
# Load the conda module
module load Anaconda3/2020.02

# Activate the 'pytorch' environment
source activate pytorch
```

Testing your PyTorch installation

To ensure that PyTorch was installed correctly, we can verify the installation by running sample PyTorch code e.g. an example from the official PyTorch getting started guide (replicated below).

Here we construct a randomly-initialized tensor:

```
import torch
x = torch.rand(5, 3)
print(x)
```

The output should be something similar to:

```
tensor([[0.3380, 0.3845, 0.3217],
        [0.8337, 0.9050, 0.2650],
        [0.2979, 0.7141, 0.9069],
        [0.1449, 0.1132, 0.1375],
        [0.4675, 0.3947, 0.1426]])
```

Additionally, to check if your GPU driver and CUDA is enabled and accessible by PyTorch, run the following commands to return whether or not the CUDA driver is enabled:

```
import torch
torch.cuda.is_available()
```

2.7.5 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.

About TensorFlow on Bede

Note: GPU must be requested in order to enable GPU acceleration by adding the flag e.g. `--gpus=1` to the scheduler command or job script. See *Running and Scheduling Tasks on Bede* for more information.

Bede has a locally installed IBM Watson Machine Learning Community Edition (WML CE) Anaconda channel that provides versions of Tensorflow, PyTorch and their dependencies especially built for the POWER architecture.

Installation in Home Directory - GPU Version

First request an interactive session, e.g. see *Running and Scheduling Tasks on Bede*.

Then GPU version of TensorFlow can be installed by the following

```
# Load the conda module
module load Anaconda3/2020.02

# Adds the local WML CE channel to the conda search path
conda config --prepend channels file:///opt/software/apps/ibm_wmlce/wmlce-1.7.0-
↪mirror/

# Create an conda virtual environment e.g. named 'tensorflow'
conda create -n tensorflow python=3.6

# Activate the 'tensorflow' environment
source activate tensorflow

# Install GPU version of TensorFlow
conda install tensorflow
```

To install a version of `tensorflow` other than the latest version you should specify a version number when running `conda install` i.e.

```
pip install tensorflow=<version_number>
```

To search for available versions use `conda search` i.e.

```
conda search tensorflow
```

Every Session Afterwards and in Your Job Scripts

Every time you use a new session or within your job scripts, the modules must be loaded and Conda must be activated again. Use the following command to activate the Conda environment with TensorFlow installed:

```
module load Anaconda3/2020.02
source activate tensorflow
```

Testing your TensorFlow installation

You can test that TensorFlow is running on the GPU with the following python code

```
import tensorflow as tf
# Creates a graph
#If using CPU, replace /device:GPU:0 with /cpu:0
with tf.device('/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))
```

Which should give the following results:

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

CHAPTER 3

Useful Training Material

The following list is a useful catalogue of training material.

A list of useful training material is also available on the gpuhackathons site:

- GPUHackathon Resources webapge

3.1 Profiling Material

- OLCF: Nsight Systems Tutorial
- OLCF: Nsight Compute Tutorial

Use the following Nsight report files to follow the tutorial.

3.2 General Training Material

- OpenACC Online Course (NVIDIA)
- NVIDIA OpenACC Advanced Training Material Container
- CUDA training at Oakridge (slides and lecture recording)
- Sheffield One or Two Day Introduction to CUDA Course (slides and labs)
- Sheffield Parallel Computing with GPUs taught module (slides, labs, lecture recordings)

CHAPTER 4

NVIDIA Profiling Tools

HPC systems typically favour batch jobs rather than interactive jobs for improved utilsation of resources. The Nvidia profiling tools can all be used to capture all required via the command line, which can then be interrogated using the GUI tools locally.

Nsight Systems and Nsight Compute are the modern Nvidia profiling tools, introduced with CUDA 10.0 supporting Pascal+ and Volta+ respectivley.

The NVIDIA Visual Profiler is the legacy profiling tool, with full support for GPUs up to pascal (SM < 75), partial support for Turing (SM 75 and no support for Ampere (SM80).

4.1 Compiler settings for profiling

Applications compiled with `nvcc` should pass `-lineinfo` (or `--generate-line-info`) to include source-level profile information.

Additionally, NVIDIA Tools Extension SDK can be used to enhance these profiling tools.

4.2 Nsight Systems and Nsight Compute

Note:

- Nsight Systems supports Pascal and above (SM 60+)
- Nsight Compute supports Volta and aboce (SM 70+)

Generate an application timeline with Nsight Systems CLI (`nsys`):

```
nsys profile -o timeline ./myapplication
```

Use the `--trace` argument to specify which APIs should be traced. See the nsys profiling command switch options for further information.

```
nsys profile -o timeline --trace cuda,nvtx,osrt,openacc ./myapplication <arguments>
```

Note: On *Bede* (Power9) the `--trace` option `osrt` can lead to `SIGILL` errors. As this is a default, consider passing `--trace cuda,nvtx` as an alternative minimum.

Once this file has been downloaded to your local machine, it can be opened in `nsys-ui`/`nsight-sys` via `File > Open > timeline.qdrep`:

Fine-grained kernel profile information can be captured using remote Nsight Compute CLI (`ncu`/`nv-nsight-cu-cli`):

```
ncu -o profile --set full ./myapplication <arguments>
```

Note: `ncu` is available since CUDA `11.0.194`, and Nsight Compute `2020.1.1`. For older versions of CUDA use `nv-nsight-cu-cli` (if Nsight Compute is installed).

This will capture the full set of available metrics, to populate all sections of the Nsight Compute GUI, however this can lead to very long run times to capture all the information.

For long running applications, it may be favourable to capture a smaller set of metrics using the `--set`, `--section` and `--metrics` flags as described in the Nsight Comptue Profile Command Line Options table.

The scope of the section being profiled can also be reduced using NVTX Filtering; or by targetting specific kernels using `--kernel-id`, `--kernel-regex` and/or `--launch-skip` see the CLI docs for more information).

Once the `.ncu-rep` file has been downloaded locally, it can be imported into local Nsight CUDA GUI `ncu-ui`/`nv-nsight-cu` via:

```
ncu-ui profile.ncu-rep
```

Or `File > Open > profile.ncu-rep`, **or** Drag `profile.ncu-rep` into the `nv-nsight-cu` window.

Note: Older versions of Nsight Compute (CUDA < `11.0.194`) used `nv-nsight-cu` rather than `ncu-ui`.

Note: Older versions of Nsight Compute generated `.nsight-cuprof-report` files, instead of `.ncu-rep` files.

4.2.1 Documentation

- Nsight Systems
- Nsight Compute

4.2.2 Training Material

- OLCF: Nsight Systems Tutorial
- OLCF: Nsight Compute Tutorial

Use the following Nsight report files to follow the tutorial.

4.2.3 Cluster Modules

- ***raplab-hackathon***:
  - `module load nvcompilers/2020`
- ***bede***:
  - `module load nvidia/20.5`

4.3 Visual Profiler (legacy)

Note:

- Nvprof does not support CUDA kernel profiling for Turing GPUs (SM75)
- Nvprof does not support Ampere GPUs (SM80+)

Application timelines can be generated using `nvprof`:

```
nvprof -o timeline.nvprof ./myapplication
```

Fine-grained kernel profile information can be genereted remotely using `nvprof`:

```
nvprof --analysis-metrics -o analysis.nvprof ./myapplication
```

This captuires the full set of metrics required to complete the guided analysis, and may take a (very long) while. Large applications request fewer metrics (via `--metrics`), fewer events (via `--events`) or target specific kernels (via `--kernels`). See the nvprof command line options for further information.

Once these files are downloaded to your local machine, Import them into the Visual Profiler GUI (`nvvp`)

- `File > Import`
- Select `Nvprof`
- Select `Single process`
- Select `timeline.nvvp` for `Timeline data file`
- Add `analysis.nvprof` to `Event/Metric data files`

4.3.1 Documentation

- Nvprof Documentation

4.3.2 Cluster Modules

- ***raplab-hackathon***:
  - `module load cuda/10.1`
  - `module load nvcompilers/2020`

- ***bede*:**
  - `module load cuda/10.1`
  - `module load cuda/10.2`
  - `module load nvidia/20.5`

CHAPTER 5

NVIDIA Tools Extension

NVIDIA Tools Extension (NVTX) is a C-based API for annotating events and ranges in applications. These markers and ranges can be used to increase the usability of the NVIDIA profiling tools.

- For CUDA `>= 10.0`, NVTX version 3 is distributed as a header only library.
- For CUDA `< 10.0`, NVTX is distributed as a shared library.

The location of the headers and shared libraries may vary between Operating Systems, and CUDA installation (i.e. CUDA toolkit, PGI compilers or HPC SDK).

The NVIDIA Developer blog contains several posts on using NVTX:

- Generate Custom Application Profile Timelines with NVTX (Jiri Kraus)
- Track MPI Calls In The NVIDIA Visual Profiler (Jeff Larkin)
- Customize CUDA Fortran Profiling with NVTX (Massimiliano Fatica)

Custom CMake `find_package` modules can be written to enable use within Cmake e.g. ptheywood/cuda-cmake-NVTX on GitHub

5.1 Documentation

- NVTX Documentation
- NVTX 3 on GitHub