OpenMind Tutorial (II): Application and Container

Shaohao Chen
Brian and Cognitive Sciences (BCS)
Massachusetts Institute of Technology (MIT)
Outline

- Environment Modules
- Anaconda for Python
- Singularity Container
  - Installation Problems and Singularity Container
  - Using Singularity on OpenMind
  - Building Singularity Images (using Vagrant VM on OpenMind)
Set Up Applications on OpenMind

- Use environment modules for software that are installed in the centralized directory on OpenMind, such as MATLAB, GCC, Singularity, CUDA, CuDNN, HDF5, etc.

- For Python packages, such as TensorFlow, Pytorch, Scikit-learn, Pandas, Keras, use either Anaconda or Singularity container.

- For many other applications, such as Julia, R, use Singularity container.
Environment Module (1)

- Use module to set up environment variables.

```
module avail    # List all available software
module avail openmind  # List the software installed for openmind
module avail mit    # List the software licensed for MIT
module load mit/matlab/2020a  # Load a specific version of a software (e.g. matlab 2019a)
module show openmind/hdf5/1.10.1  # Show the environment variables for a software (e.g. hdf5)
module list       # List currently loaded software
module remove openmind/hdf5/1.10.1  # unload a software
module switch mit/matlab/2020a mit/matlab/2019a  # Switch a software to a different version
module purge     # Unload all current loaded software
module use /path/to/modulefiles  # Use a module in a nonstandard location (for user-defined modules)
```
Load the modules:

<table>
<thead>
<tr>
<th>Module Load Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module load openmind/gcc/5.3.0</td>
<td># GNU compiler for C/C++ and Fortran</td>
</tr>
<tr>
<td>module load mit/matlab/2020a</td>
<td># A multi-paradigm numerical computing software</td>
</tr>
<tr>
<td>module load openmind/freesurfer/6.0.0</td>
<td># A software for Brain MRI analysis</td>
</tr>
<tr>
<td>module load mit/mathematica/10.3.1</td>
<td># A technical computing software</td>
</tr>
<tr>
<td>module load openmind/cuda/10.1</td>
<td># For GPU programming and applications</td>
</tr>
<tr>
<td>module load openmind/cudnn/10.1-7.6.4</td>
<td># A DNN lib optimized for GPUs</td>
</tr>
<tr>
<td>module load openmind/hdf5/1.10.1</td>
<td># A library for fast or parallel I/O</td>
</tr>
<tr>
<td>module load openmind/singularity/3.2.0</td>
<td># A container for HPC</td>
</tr>
</tbody>
</table>

Then the libs are available for building or running software.

Executables will be available, for example:

```
matlab -nodesktop # Open Matlab without GUI
```
Anaconda for Python (1)

- **Method 1: Load an Anaconda module**

  - Several versions of Anaconda are installed in a centralized location (/cm/shared/openmind):

    ```
    module load openmind/anaconda/3-2019.10  # A full anaconda distribution
    module load openmind/miniconda/2020-01-29-py3.7  # A basic Anaconda platform
    ```

  - Start Python and import your desired packages:

    ```
    $ python
    Python 3.7.4 (default, Aug 13 2019, 20:35:49)
    [GCC 7.3.0] :: Anaconda, Inc. on linux
    Type "help", "copyright", "credits" or "license" for more information.
    >>> import torch
    ```

  - Some widely-used Python packages have been installed. If your desired python package is not available, submit an issue to request for it, or see the next slides for user customized installations.
Method 2: Install Anaconda in your own directory

- Go to a directory with a large quota

- Download an installer. See all versions here: [https://repo.anaconda.com/archive/](https://repo.anaconda.com/archive/). An example:

  ```bash
  cd /om2/user/$USER
  wget https://repo.anaconda.com/archive/Anaconda3-2019.10-Linux-x86_64.sh
  mkdir /om2/user/$USER/anaconda
  sh Anaconda3-2019.10-Linux-x86_64.sh -f -p /om2/user/$USER/anaconda
  ```

- Install on /om2 instead of /home

- Run the installer.

- Then install your desired Python packages using `conda install`.

  ```bash
  conda init bash
  ```

  Set up conda environment in the ~/.bashrc file for bash shell.

  ```bash
  source ~/.bashrc
  ```

  Set up the environment in the current shell. Or, log out and log in again.

  ```bash
  conda install tensorflow-gpu
  ```

  Install Tensorflow with GPU support

  ```bash
  conda install pytorch-gpu
  ```

  Install Pytorch with GPU support

- Then the packages will be available in the base channel.
Anaconda for Python (3)

- Manage packages using channels, for example,

```
conda create -n tf  # Create a channel for Tensorflow.
conda activate tf   # Activate the channel.
conda install tensorflow-gpu # Install Tensorflow in the channel.
conda deactivate  # Deactivate the channel.
conda create -n torch # Create a channel for Pytorch.
conda activate torch # Activate the channel.
conda install pytorch-gpu # Install Pytorch in the channel.
```


- **Notes:** Anaconda provides software stack in its own world. Using `pip install` in Anaconda may or may not work for some packages.
Installation Problem in Linux

- On HPC clusters, usually users do not have root privilege, and thus are not allowed to use *apt* or *yum* for automatic installation.

- It is almost always possible to build a software from source code without the need of root privilege, but this can be time-consuming sometimes, for example:
  - There are many nested dependencies.
  - Only specific versions of dependencies are compatible with the target.
  - Need to substantially modify configure file or Makefile to meet requirements.

- Hard to maintain too many versions of software.

- Many precompiled programs conflict with the host operating system (known as the *glibc* error).
Container

- **Container**: Provides an additional operating system and software stack within the host system. Enables full access to the hardware on the host computer.
- **An operating system within an operating system** (like a VM, but different).
- **Need root privilege to build** a container on a host computer.
- Users can select the needed **operating system**, install all **dependencies**, and build the whole **software stack**. (Automatic build tools **apt** or **yum** are allowed.)
- **Run the container without root privilege** on the host system.
- **Portability**: Run the container on any platform.

- **Container image**: a file that contains everything needed to run applications.
- **Mobility**: Easy to transfer a container image.
Singularity

- **Docker container**: for local laptop/desktop or enterprise cloud computing platforms.
- **Singularity container**: for HPC clusters.
  - Has built-in support for **GPU** and **MPI**.
  - Supports **InfiniBand** and **Lustre**.
  - Works well with **resource managers** (e.g. Slurm, SGE, Torque, etc.)
  - As easy as installing a single package onto the host operating system. Singularity is installed as a module on OpenMind.

- Docker image is compatible with Singularity and it is easy to convert it to a Singularity image.
- There are lots of container images available (e.g. **Docker Hub**, **Singularity Hub**, **Nvidia GPU Cloud**, or **Neurodocker**).
When to use Singularity on OpenMind?

- **Python packages** (especially from Pypi or Github):
  
  Download a Singularity image for your desired Python packages.
  
  Build your desired Python packages using `pip install` in a Singularity container.

  *Anaconda/Minicoda is an alternative solution for many Python packages.*

- **Software in an Ubuntu operating system:**
  
  Download a Singularity image that supports Ubuntu, then use the `apt-get` installation tool.

- **GPU programs:**
  
  Download a Singularity image with NVidia software (such as CUDA, CuDNN).

- **Executables precompiled in systems other than CentOS 7 (source codes unavailable):**
  
  For example, precompiled C/C++, Fortran or MATLAB programs.

- **Other software that is hand to install:**
  
  For example, Julia, R, MPI programs, chemistry packages (such as NAMD).
A typical Singularity Workflow on Openmind

1. Load a Singularity module. For example,

   module load openmind/singularity/3.2.0

2. Download a Singularity image that provides desired operating system and software stack (using `singularity pull` or `singularity build`). No installation is required in many cases. If necessary, build a user-customized Singularity image (using `sudo singularity build`).

3. Write a run script including environment variables and job steps. It is better to test the job steps in a Singularity shell (using `singularity shell`).

4. Execute the run script in the container (using `singularity exec` or `singularity run`).

5. Write a Slurm batch job script to include steps 1 and 4, then submit the job script (using `sbatch`).
Download Singularity Images

• **Pull:** download and do not convert

  singularity pull docker://python        # Download a Docker image from Docker Hub.
  singularity pull docker://python:latest # Download a specific version.
  singularity pull --name python-latest.sif docker://python:latest # Download and rename.

• **Build:** download and convert it to a Singularity image (in the default squashfs format).

  singularity build python-numpy.simg docker://adreeve/python-numpy          # Python with numpy
  singularity build ubuntu18-gpu.simg docker://nvidia/cuda:10.1-cudnn7-devel-ubuntu18.04  # Ubuntu with CUDA

• **To avoid overusing home directory:** execute build or pull at /om2, and redirect cache data:

  export SINGULARITY_CACHEDIR="/om2/user/$USER/.singularity"
Singularity Shell

- **Shell:** Login the shell on the container:

  ```
  singularity shell /cm/shared/singularity-images/python-numpy.simg
  ```

- Explore the environment within Singularity shell:

  ```
  > pwd  # The current directory: the same as the host directory where singularity was executed.
  > ls ~ # List the home directory: the same as the host home directory
  > ls / # List the root directory, different from the host root directory
  > whoami # The same user name as on the host
  > groups # The same Linux groups as on the host
  > cat /etc/os-release # Check Linux system info: different from the host
  > cat /proc/cpuinfo # Check CPU info: the same as the host
  ```

- Test an application (e.g. Python with Numpy) within the Singularity shell:

  ```
  > python # Launch python
  >>> import numpy as np # Import Numpy in python interface
  ```
Bind Paths and Environment Variables

• Default bind directories: $PWD, $HOME, /tmp, /proc, /dev, /sys

• Bind directories with `-B` or `--bind`:

```bash
singularity shell -B /om2 python-numpy.simg        # Bind one directory
singularity shell -B /om,/om2 python-numpy.simg    # Bind multiple directories
singularity shell -B /scratch:/tmp python-numpy.simg # Bind a directory and rename it
```

• Alternatively, set the environment variable `SINGULARITY_BINDPATH`:

```bash
export SINGULARITY_BINDPATH="/om2,/om,/scratch:/tmp"    # Bind multiple directories
singularity shell python-numpy.simg                   # Shell into the container
```

• Pass runtime environment variables from the host to the container:

```bash
SINGULARITYENV_PATH=$PATH singularity shell python-numpy.simg  # Pass the variable PATH
```

• If building a container from recipes, the environment variables can be set at the section `%environment`. 
Singularity Exec

**Exec:** execute a command in the container.

```
singularity exec python-numpy.simg pwd  # Execute the command pwd
singularity exec -B /om2 python-numpy.simg ls /om2  # Check the bound directory.
SINGULARITYENV_PREPEND_PATH=$PATH singularity exec python-numpy.simg echo $PATH  # Check passed variable
singularity exec python-numpy.simg python arange.py 10  # Execute a python program
singularity exec python-numpy.simg cat /singularity  # Print the default “runscript” in the container.
```

**An example code** (named as `arange.py`) to create a Numpy array:

```
import sys  # Import system package
import numpy as np  # Import numpy package
n = int(sys.argv[1])  # Use the first input argument from command line
a=np.arange(n)  # Create a numpy array sized n
print(a)  # Print the result
```
Singularity Run

• Run: run the default “runscript” in the container (with input arguments).

• What `singularity run` would return depends on the runscript (made by the builder).

• For example,

```bash
singularity run python-numpy.simg python arange.py 10 # The same as singularity exec here.
singularity run tf-latest-gpu.simg # Shell into the container with pre-customized environment
```

• Typically, a line `exec "$@"` is included in the runscript, which executes the arguments as a command.

• If the image is built from recipes, the runscript can be defined in the section `%runscript`. If the runscript is not defined, it is default to run `/bin/bash`, meaning shell into the container.

• Since the outcome of `singularity run` is uncertain, it is more recommended to use `singularity exec` to execute a self-defined run script to run your applications.
An Example for Running Tensorflow

• Download an image with GPU-enabled Tensorflow

```bash
singularity build tf-latest-gpu.simg docker://tensorflow/tensorflow:latest-gpu  # Download and convert
```

• Download a Tensorflow example: MNIST

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/mnist_with_summaries.py

• Run a Tensorflow program with GPU using Singularity (using the `-nv` flag):

```bash
srun -t 02:00:00 --gres=gpu:1 --constraint=any-gpu --pty bash  # Interactive session with GPU
module load openmind/singularity/3.2.0  # Load Singularity module
singularity exec --nv /cm/shared/singularity-images/tf-latest-gpu.simg nvidia-smi  # Check available GPUs (optional)
# Run the program
singularity exec --nv /cm/shared/singularity-images/tf-latest-gpu.simg python mnist_with_summaries.py
```
Use Singularity in Batch Job

- A sample Slurm batch job script to run a Tensorflow program using Singularity:

```bash
#!/bin/bash
#SBATCH -t 01:30:00 # walltime = 1 hours and 30 minutes
#SBATCH -N 1 # one node
#SBATCH -n 2 # two CPU (hyperthreaded) cores
#SBATCH --gres=gpu:titan-x:1 # one NVIDIA Titan X GPU
module load openmind/singularity/3.2.0 # load singularity module

singularity exec --nv /cm/shared/singularity-images/tf-latest-gpu.simg python mnist_with_summaries.py
```

- If there are multiple job steps running in the Singularity container, write a bash script to include all job steps. Make it executable.  

```bash
chmod +x run.sh
```

- Then execute it in the batch job script:

```bash
singularity exec --nv /cm/shared/singularity-images/tf-latest-gpu.simg ./run.sh
```
Use Jupyter Notebook in Singularity

- **Jupyter Notebook**: A popular IDE for Python programmers. Work interactively.
- **Jupyter notebook can be used without Singularity.**
- **Use Jupyter notebook in Singularity to support desired python packages.**
- First, download a Singularity image with the desired packages.
- **Do not use Jupyter notebook on the head node. Use it on a compute node instead.** Request an interactive session, then:

```bash
unset XDG_RUNTIME_DIR  # Unset this conflicted variable.
singularity exec --nv name.simg jupyter-notebook --ip=0.0.0.0 --port=9000  # Fill in: image name, port number.
```

- In a terminal window on your local computer, do ssh tunneling:

```bash
ssh -L xxxx:nodeXXX:9000 username@openmind7.mit.edu  # Fill in: xxxx (>1024), node number XXX, user name.
```

- Open a local browser. Input address *localhost:xxxx*, where xxxx is the same 4 digits as above.
Build Singularity Containers

- Three ways to build container images.
  - On your MAC or Windows laptop/desktop, build a docker image on it, then upload to Openmind.
  - If you have a laptop/desktop with Linux OS, install Singularity on it, then build Singularity images.
  - On OpenMind, use a Vagrant VirtualBox in which Singularity is installed.

- Two ways to build Singularity images:
  - Build interactively
  - Build from a recipe file

- Singularity Registry: a simple cloud solution for building, storing and sharing your container.
Use Vagrant VirtualBox (1)

• **Use Vagrant VirtualBox to provide root privilege on Openmind.**

• **Vagrant VirtualBox:** a tool for building and managing virtual machine environments.

• Look for desired virtual boxes (in which Singularity has been installed) on **Vagrant Cloud:**
  [https://app.vagrantup.com/boxes/search](https://app.vagrantup.com/boxes/search)

• To avoid overuse home directory: redirect default directories.

```bash
export VAGRANT_HOME=/om2/user/$USER/.vagrant.d  # Redirect Vagrant system files
ln -s /om2/user/$USER/.vagrant.d ~/.vagrant.d       # Another way to redirect Vagrant system files
ln -s /om2/user/$USER/.VirtualBox ~/VirtualBox\ VMs # Redirect VirtualBox files
mkdir -p /om2/user/$USER/vagrant                  # Create a directory under /om2 for Vagrant work
 cd /om2/user/$USER/vagrant                       # Will start vagrant work under /om2
```

• **Do not run Vagrant on the head node.** Request an interactive session:

```bash
srun -t 03:00:00 --mem=10G --pty bash  # Request sufficient time and memory
```
Use Vagrant VirtualBox (2)

- **Init**: create the Vagrantfile.

```bash
vagrant init sylabs/singularity-3.2-ubuntu-bionic64  # The name is obtained from Vagrant Cloud
```

- Customize the Vagrantfile:

```ruby
config.vm.provider "virtualbox" do |vb|
  vb.memory = "10000"  # Assign 10 GB RAM (default 1G). Will be sufficient for building Singularity container.
end
```

- Launch Vagrant VirtualBox (in the directory containing the Vagrantfile):

```bash
vagrant up  # Launch Vagrant. Will install the VirtualBox for the first time.
vagrant ssh  # Login the Vagrant VirtualBox
```

- In the VirtualBox:

```bash
which singularity  # Check whether singularity is available.
cd /vagrant  # Shared directory: /vagrant in VirtualBox = the directory containing Vagrantfile on the host
```
Build Singularity Container Interactively

• Download a base image (e.g. on Docker hub or Singularity hub) that has the desired operating system and as many dependencies as possible.

• The Sandbox format is convenient for accessing interactively.

  ```
singularity build --sandbox <image-name> <address_of_the_base_image>
  ```

• Shell into the Sandbox with the --writable option and then make changes as needed,

  ```
sudo singularity shell --writable <image-name>
  > # make your desired changes now (e.g. install packages for Python, R, Julia, or other applications.)
  > exit
  ```

• Convert Sandbox format to Squashfs format,

  ```
sudo singularity build <image-name>.simg <image-name>
  ```
An Example for Building Pytorch

- Download a base image that has Ubuntu, CUDA and CuDNN:

```bash
singularity build --sandbox ubuntu18-gpu docker://nvidia/cuda:10.1-cudnn7-devel-ubuntu18.04
```

- Build the desired packages interactively:

```bash
sudo singularity shell --writable ubuntu18-gpu
> apt-get update -y  # Update latest packages in Ubuntu.
   # Install python3, pip3, and python3-tk (required by matplotlib) without user prompts.
> DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends python3 python3-tk python3-pip
> pip3 install torch torchvision numpy matplotlib  # Install Python modules.
> exit
```

- Convert Sandbox format to Squashfs format,

```bash
sudo singularity build pytorch-gpu.simg ubuntu18-gpu
```
An Example for Running Pytorch

• Download a Pytorch example for training neural network:
  
  https://pytorch.org/tutorials/beginner/examples_nn/two_layer_net_n.pdf

• A job script (e.g. named job.sh) to run the Pytorch program using Singularity:

```bash
#!/bin/bash  # Bash shell
#SBATCH -t 01:00:00  # walltime = 1 hour
#SBATCH -n 2  # two CPU (hyperthreaded) cores
#SBATCH --gres=gpu:1  # one GPU of any type
#SBATCH --constraint=high-capacity  # any type of GPU on the cluster
module load openmind/singularity/3.2.0  # load a singularity module
singularity exec --nv /cm/shared/singularity-images/pytorch-gpu.simg python3 torch.nn.py  # Run the program
```

• Submit the job script

  sbatch job.sh
Build from a Singularity Recipe File

- A singularity recipe file is used to customize operating system, applications, and environment.
- An example for GPU-enabled Pytorch in Ubuntu:

```
Bootstrap: docker  # Download from Docker hub
From: nvidia/cuda:10.1-cudnn7-devel-ubuntu18.04  # Base image: Utuntu system with CUDA and CuDNN
%post  # Section name
  apt-get update -y  # Update latest packages in Ubuntu.
  # Install python3, pip3, and python3-tk (required by matplotlib) without user prompts.
  DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends python3 python3-tk python3-pip
  rm -rf /var/lib/apt/lists/*  # Delete useless packages.
  pip3 install torch torchvision numpy matplotlib  # Install needed Python modules.

%post: Execute commands after the base OS has been installed, including making directories, and installing software and libraries.

Build from the recipe:  sudo singularity build image-name.simg recipe-name.def
```
More on Recipe Files (1)

• **%setup**: Execute commands (e.g. copying files) after header and before %post.

• **%files**: Copy files from host to container after header and before %post (for versions >2.3)

```bash
%setup
    touch file  # Create a file
%file
    /path/to/host/file  /path/to/container/file  # Use the convention of the cp command
```

• **%environment**: Set environment variables for runtime (not for build time).

```bash
%environment
    VAR=value  # Runtime environment variables
%post
    echo ‘export VAR=value’ >> $SINGULARITY_ENVIRONMENT  # Build-time and runtime environment variables
```

```bash
singularity exec pytorch-gpu.simg | env  # Print runtime environment variables.
```
• **%label**: Define user-defined labels (e.g. name, version). Printed by `singularity inspect`.

• **%help**: Print help message. Printed by `singularity help`.

• **%runscript**: Define commands that will be executed by `singularity run`.

```plaintext
%runscript
exec "$@"  # Treat the input arguments as a command and execute it.
```

```
singularity run --nv pytorch-gpu.simg python3 train-xor.py  # The same as exec in this case.
```

• **%test**: Define commands for test run after build. Build with `--notest` to avoid test when necessary, for example, when building a GPU-enabled container on a non-GPU node.

```plaintext
sudo singularity build --notest pytorch-gpu.simg pytorch-gpu.def  # Build with no test run.
```
More on Recipe Files (3)

• When necessary, build a single container with two or three different apps that each has their own runscripts and custom environments.

• `%appinstall`: commands to install the app (similar to `%post` but only for one app here).

• `%appfiles`: copy files after header and before installing the app.

• `%apprun`: runscript for the app.

• `%appenv`: run time environment variables for the app.

Best practice for building from recipes:

• Download an base container image that provides most desired software, then install additional software as needed.

• Download a recipe file on internet (e.g. from Github), use it directly.
More on Singularity Container

- Check metadata in a Singularity container

```bash
ls /singularity.d     # List the metadata folder in the Singularity container
cat /singularity.d/Singularity   # The build recipe file
cat /singularity.d/env/90-environment.sh    # Environment variables in %environment
cat /singularity.d/env/91-environment.sh    # Environment variables in $SINGULARITY_ENVIRONMENT
cat /singularity.d/runscript        # The default run script. The /singularity is a soft link to this file
cat /singularity.d/labels.json      # The labels
ls /singularity.d/libs            # The host-system libraries that are mapped into the container
```

- All `*.sh` files in the `env` directory are sourced in alpha-numeric order when the container is initiated. The later will overwrite the former.
- The `--nv` flag maps NVIDIA libs into the container to enable usage of GPU.
Further Help

- **OpenMind documentations on MIT Github**: [https://github.mit.edu/MGHPCC/OpenMind/wiki](https://github.mit.edu/MGHPCC/OpenMind/wiki)
- **Get started**: [https://github.mit.edu/MGHPCC/OpenMind/wiki/Getting-started](https://github.mit.edu/MGHPCC/OpenMind/wiki/Getting-started)
- **OpenMind website (including tutorial slides and videos)**: [https://openmind.mit.edu/](https://openmind.mit.edu/)
- **Submit an issue**: [https://github.mit.edu/MGHPCC/openmind/issues](https://github.mit.edu/MGHPCC/openmind/issues)

  - “Not watching”: Be notified when participating or @mentioned.
  - “Watching”: Be notified of all conversations. To avoid a deluge of emails, set up email filtering.

  If you know the answer or have seen the problem before, consider replying it. We need communication between users to keep it functioning optimally.

- **Slack channel for OM users**: openmind-46.slack.com
- **User meetings**: Walk-in for short-time questions. Make an appointment for discussions.
- **Group-rep meetings**: Monthly. Communicate with your group reps.
- **Email**: openmind-help@mit.edu . **Office**: 46-4115D.