User guide for the Kepler cluster of the ARI

Fabian Klein

24th July 2016

1 Kepler cluster

1.1 Configuration

1.2 Kepler Hardware

Each Kepler node is equipped with two 8 core Intel Xenon E5-2650 processors. One of them is assigned to the CPU queue, while the other one is assigned to the GPU queue. Hence, there are 8 threads available for the CPU queue(8 devices) and 8 more for the GPU queue(1 device). Hence, you can use e.g. 8 OpenMp threads atop on one MPI thread using 1 GPU. The GPU is one K20M(rev a1) per node.

1.3 Organisation of diskspace

There are two general parts of the diskarray. The "home" and the "work" part. "home" is backed up, whereas "work" is not. Usually, you will get space on home and a more bulky folder on "work" on request.

1.4 Usage

1.4.1 Queueing system

Most important commands:

sbatch *scriptname*	Submit *scriptname* script
scancel *jobID*	Cancel job *jobID*(Must be your own job)
squeue	Display current state of queues(called partitions)

Two scripts for use with "sbatch" have been supplied to you already. Here are some more details on their contents and the commands.

Command	Effect	Comments
`#!/bin/bash`	Makes it a valid bash script	
`#SBATCH -J <jobname>`	Sets the job's name	
`#SBATCH -p <partition>`	Chooses partition(often also called "queue")	Choices `CPU`/`GPU`
`#SBATCH --nodes=N`	Number of nodes	$12 \leq N \leq 1$
`#SBATCH --gres=gpu:1`	Request one GPU per Node	Only in GPU queue!
`#SBATCH --ntasks-per-node=n`	Number of tasks per node	$8 \leq n \leq 1$
`#SBATCH --cpus-per-task=m`	Number of threads per task(OpenMP)	$mn \leq 8$
`#SBATCH --time=xx:yy:zz`	Desired runtime(Killed afterwards)	$\leq 72:00:00$
`%J`	Access for jobnumber	
`#SBATCH --output=outname.%J.out`	Redirect Stdout to filename	
`#SBATCH --error=errorname.%J.err`	Redirect Stderror to filename	

Remember to load any modules that you might need and do any exports after this part of script! See the loading of cuda 5.0 in the GPU script and the export of the OpenM as also shown here. For details on modules see 1.5.

```
module load cuda/5.0

if [ -n "$SLURM_CPUS_PER_TASK" ]; then
  omp_threads=$SLURM_CPUS_PER_TASK
else
  omp_threads=1
fi
export OMP_NUM_THREADS=$omp_threads
```

If `--cpus-per-task` is not defined it defaults to 1 in this setup. After that you can start your program with `mpirun -np M ./Yourprogram`, where $M = nN$. For your convenience the example scripts calculates that for you via.

```
mpiprocs=$(($SLURM_NTASKS_PER_NODE * $SLURM_NNODES))
mpirun -np $mpiprocs ./yourprogram
```

1.5 Module system

The Kepler cluster now contains a module system in order to make different compilers, programs and e.g. CUDA available to users. The following commands are most important for the `module` package.

- `module avail` List available modules you can currently load
- `module load <modulename>` Load module specified by modulename
- `module list` List currently loaded modules
- `module unload <modulename>` Unload specified module
- `module switch <current loaded module> <switchable module>` Switch a currently loaded module with a module eligible to be switched with it

The `load`, `unload` and `switch` allow you to use the tab completion. Be aware that their are dependencies and excluding relations betwen modules. You should either check `module avail` or use tab completion. If you always need certain modules loaded please add the command add the end of your `.bashrc`.

1.6 Using (different versions of) CUDA

On the Kepler cluster there is a choice of CUDA versions for you to use. This tutorial will show you how to make usw of the different versions.

1.6.1 Currently available CUDA versions

All CUDA versions are located in /opt/local. As of now (05.02.2016) the CUDA versions displayed in table 1 are installed on Kepler. If you need a version not installed on the cluster please notify the admins.

CUDA Version	exact location
4.2	/opt/local/cuda42
5.0	/opt/local/cuda50
7.0	/opt/local/cuda-7.0

Table 1: Table displaying currently installed CUDA Versions

1.6.2 Switching between versions

CUDA is managed by a module system on the Kepler cluster. You can use any of the three choices:

- `module load cuda/4.2`
- `module load cuda/5.0`
- `module load cuda/7.0`

in order to load the desired version of CUDA into your path/library path variable. You can switch between version by using the `module switch` command as explained in subsection 1.5. You need to edit your makefiles and library addtions such that is matches the CUDA paths.

2 Getting your own python using pyenv

3 Installation of pyenv

3.1 Installation and updates

This is mainly extracted from `https://github.com/yyuu/pyenv`.
Initially you need to pull the git repository of pyenv into /.pyenv. You do this by:
`git clone https://github.com/yyuu/pyenv.git ~/.pyenv`
Git then proceeds to download the current version from gitHub. You then need to add some things to your /.bashrc by:

```
echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(pyenv init -)"' >> ~/.bashrc
```

Then you need to get the new variables into the shell. The savest way to do so is to log out and log in again.
If you want to update to a newer version of pyenv at some point you just execute the commands

```
cd ~/.pyenv
git pull
```

3.2 Usage

If everything worked correctly you can now use pyenv anywhere in your folders. `pyenv commands` lists all commands of pyenv. For a detailed description see `https://github.com/yyuu/pyenv/blob/master/COMMANDS.md`. The most important ones are listed below.

pyenv install –list	Lists all possible versions of python which can be installed
pyenv install version	Installs python version. E.g. pyenv install 2.7.6.
pyenv rehash	Execute this after every installation to be save
pyenv uninstall version	
pyenv local version	Sets local python version (folder and below) to version
pyenv global version	Set global python version

Hint: `pip` already comes with python for python >= 2.6.6. For older versions you have to build it first (not documented here). You can now install new python packages to the currently used python distribution by

```
pip install <packageName>
```

If you want to upgrade a package use `pip install --upgrade SomePackage`. Searching for packages:

```
pip search package name
```

See `http://www.pythonforbeginners.com/basics/how-to-use-pip-and-pypi` for a full list. It should be noted that some packages (e.g. scipy) need the installation if certain system packages. Please notify an administrator that you need the certain package and it will be installed if possible.

3.2.1 Manually change the python version used in a specific folder