Setting Up the Private Fugaku Environment

Follow these steps to set up an environment where you can run Private Fugaku:

1. Create an AWS EC2 Instance
2. Install ParallelCluster
3. Create a Definition File to Create a Cluster with ParallelCluster
4. Create the Cluster
5. Install Singularity
6. Run Applications

Note: The creation of the AWS EC2 instance is performed by the user. The steps from 2 onwards are explained below.

2. Installing ParallelCluster

Refer to the AWS Document:

- 「Install AWS ParallelCluster in a virtual environment (recommended)」
- https://docs.aws.amazon.com/parallelcluster/latest/ug/install-v3-virtual-environment.html

3. Creating a ParallelCluster Definition File

Refer to the AWS Document:

- 「Cluster configuration file」
- https://docs.aws.amazon.com/parallelcluster/latest/ug/cluster-configuration-file-v3.html
- https://docs.aws.amazon.com/parallelcluster/latest/ug/Scheduling-v3.html#Scheduling-v3-SlurmQueues

4. Creating the Cluster

Refer to the AWS Document

- 「Configure and create a cluster with the AWS ParallelCluster command line interface」
- https://docs.aws.amazon.com/parallelcluster/latest/ug/install-v3-configuring.html

5. Installing Singularity

Refer to the SingularityCE Admin Guide

- 「Install from Source」
- https://docs.sylabs.io/guides/4.1/admin-guide/installation.html#install-from-source

6. Running Applications

- Download the Singularity container image from the Sylabs Cloud Library:

```
$ singularity pull library://riken-rccs/virtual-fugaku/vf-ver1.2
```

- This container includes the following frequently used applications for Fugaku:
  - autodock-vina
  - cp2k
  - cpmd
  - darshan-runtime
  - fds
  - ffmpeg
  - frontistr
  - genesis
  - gnuplot
  - grads
  - gromacs
  - gsl
  - julia
  - Lammps
  - openbabel
  - openfoam
  - openfoam-org
  - openmx
  - paraview
  - petsc
  - povray
  - py-ase
  - py-matplotlib
  - py-mpi4py
  - py-netcdf4
  - py-pandas
  - py-scikit-learn
  - py-scipy
  - py-tensorflow
  - py-toml
  - py-torch
  - py-xarray
  - quantum-espresso
  - salmon-tddft
  - scale
  - tmux
  - wrf

Note: Examples of running genesis, gromacs, and scale are shown in the following slides.

Example 1: Running genesis

- The condition for completion is that the profile information of the measurement time is output in the standard output "Output_Time> Averaged timer profile (Min Max)".
- Input Data:
  https://www.r-ccs.riken.jp/labs/cbrt/wp-content/uploads/2020/12/benchmark_mkl_ver4_nocrowding.tar.gz
- Execution Script:

```
#!/bin/bash
#SBATCH -p satf01
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH -J test_genesis

SIFFILE=~/vf-ver1.2_latest.sif

export SINGULARITY_BIND=${PWD},/opt/amazon,/usr/lib64/libefa.so.1,/usr/lib64/libibverbs.so.1
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

cd npt/genesis1.6_2.5fs/jac_amber
mpiexec --use-hwthread-cpus -n ${SLURM_NTASKS} singularity run ${SIFFILE} spdyn p${SLURM_NTASKS}.inp
```

- Execution Result: Normal Operation

```
Output_Time> Averaged timer profile (Min, Max)
  total time      =      21.991
    setup         =       0.852
    dynamics      =      21.138
      :
```

Example 2: Running gromacs

- The condition for completion is that the message "Finished mdrun on rank 0" with the date and time is output at the end of md.log.
- Input Data:
  https://ftp.gromacs.org/pub/benchmarks/ADH_bench_systems.tar.gz
- Execution Script:

```
#!/bin/bash
#SBATCH -p satf01
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH -J test_gromacs

SIFFILE=~/vf-ver1.2_latest.sif

export SINGULARITY_BIND=/opt/amazon,/usr/lib64/libefa.so.1,/usr/lib64/libibverbs.so.1

mpiexec --use-hwthread-cpus -n 1 singularity run ${SIFFILE} gmx_mpi grompp -f pme_verlet.mdp -c conf.gro -p topol.top -o ions.tpr
mpiexec --use-hwthread-cpus -n ${SLURM_NTASKS} singularity run ${SIFFILE} gmx_mpi mdrun -ntomp ${SLURM_CPUS_PER_TASK} -s ions.tpr
```

- Execution Result: Normal Operation

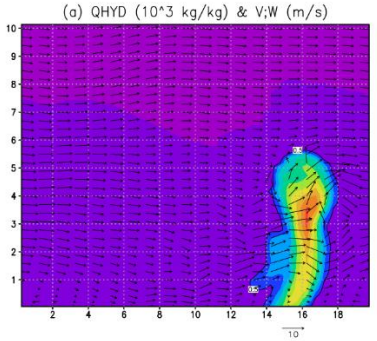
```
               Core t (s)   Wall t (s)        (%)
       Time:     1825.922       57.060     3200.0
                 (ns/day)    (hour/ns)
Performance:       30.287        0.792
```

Example 3: Running scale

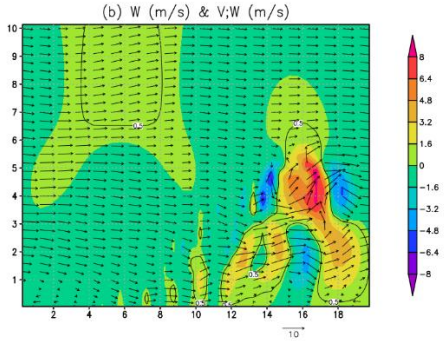
- The condition for completion is that the image file created with GrADS matches Figure 3.1.1 in the USERS GUIDE based on the execution result.
- Input Data:
  https://scale.riken.jp/archives/scale-5.4.5.tar.gz
- Execution Script:

```
#!/bin/bash
#SBATCH -p satf01
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH -J test_scale

SIFFILE=~/vf-ver1.2_latest.sif

export SINGULARITY_BIND=${PWD},/opt/amazon,/usr/lib64/libefa.so.1,/usr/lib64/libibverbs.so.1

 cp -pr ../scale-5.4.5/scale-rm/test/tutorial/ideal/* .

#初期値の作成
 cp sample/init_R20kmDX500m.conf ./init_R20kmDX500m.conf
 mpiexec --use-hwthread-cpus -n ${SLURM_NTASKS} singularity run ${SIFFILE} scale-rm_init
init_R20kmDX500m.conf

#シミュレーションの実行
 cp sample/run_R20kmDX500m.conf ./run_R20kmDX500m.conf
 mpiexec --use-hwthread-cpus -n ${SLURM_NTASKS} singularity run ${SIFFILE} scale-rm
run_R20kmDX500m.conf

#後処理
 cp sample/net2g_R20kmDX500m.conf ./net2g_R20kmDX500m.conf
 mpiexec --use-hwthread-cpus -n ${SLURM_NTASKS} singularity run ${SIFFILE} net2g
net2g_R20kmDX500m.conf
```

- Execution Result: Normal Operation



ideal_qhyd.png



ideal_W.png