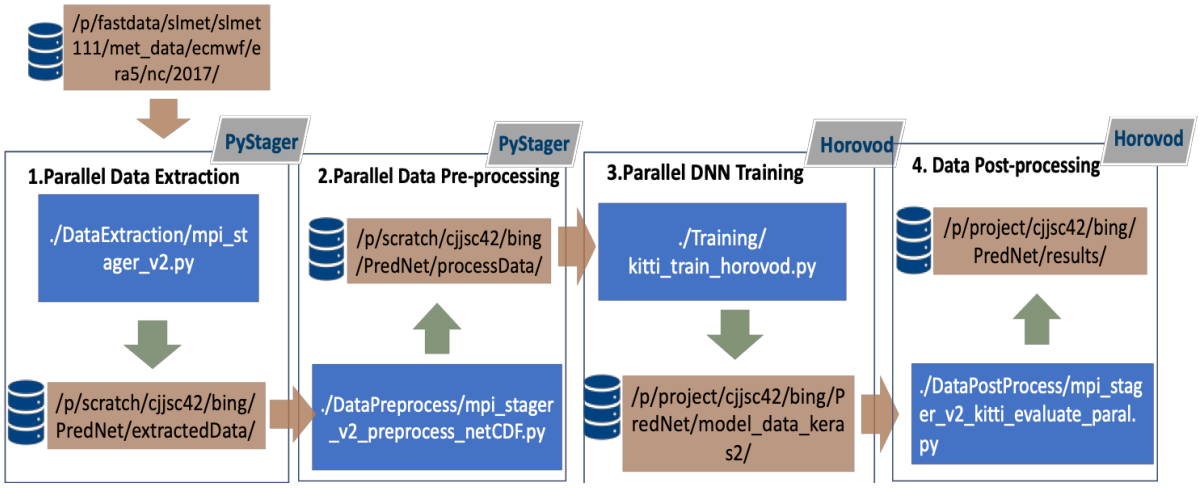
ESiWACE2
Container Hackathon for Modellers

Application Description

This project implements a workflow for parallel deep learning to predict the 2m temperature based on one master student master thesis in JSC.

The study focuses on applying data-driven deep learning methodologies to the field of weather forecasting, specifically air temperature over Europe. A future frame prediction model from the computer vision field is trained with the three input variables air temperature, surface pressure, and the 500 hPa geopotential in order to predict the air temperature itself. Future frame prediction models generate the next frame(s) from a given number of preceding frames.

The workflow consists of a sequence of steps (Data Extraction, Data Preprocessing, Training and Data Postprocess) to implement video prediction, and in each step try to Parallel for accelerating the whole prediction process.



Containerisation Approach

Setting up a container individually in 3 different machines (1 Mac and 2 Linux)
We have decided to containerizing two 2 tasks, both different parts of the Workflow for frame prediction. 1. training and 2. pyStager
Bing and Jan will work on Task 1 and Amirpasha will work on Task 2.

- Explain the starting point (e.g., CPU-code compiled with XXX compiler, GPU access using CUDA, etc)
- Shortly describe the test case that verifies the code is functioning correctly
- Steps you made porting the code to a Docker container (e.g., which components ported)

- OPTIONAL Performance profile of the code running natively and from a container, e.g., speedup graph.

Day 1 (03/12/2019) Summary

After the break, we are starting a docker-file for each of the tasks.
Starting a Dockerfile for the Task1 and Task 2, by building a base.
We have created the docker-file based on the need of the project.

Task 1:
After setting up Docker on all machines and getting familiar with its usage we tried to cointainerize our Python application.
As we are using Tensorflow, Keras and Horovod for the Python script we used a pretty complete base image already including Horovod and all its dependencies.
We built the container image with
`docker build -t wfppdl/parallel_training:v1.0 -f ../docker/Dockerfile_parallel ../docker` in the source code directory.

However, that image was designed to work with GPUs and as we dont have GPUs on our Laptops so the script did not work in the Container as it was looking for GPUs (or actually for Cuda on the host system) which didnt exist.

Task 2:
We have executed the docker images with
`sudo docker images`

Change the entrypoint of the docker :
sudo docker run -ti --entrypoint "sh" wfppdl/pystager:v1.0

We bound a mount to the docker image (https://docs.docker.com/storage/bind-mounts/)

```
$ docker run -d \
  -it \
  --name devtest \
  --mount type=bind,source="$(pwd)"/target,target=/app \
  Nginx:latest
```

Task 2 is able to be run on the docker.

Shared Notes...
Containerization
Short description of how a docker image works.
In container : the container is setting on top of the kernel
VM: hypervisor is running on top of teh kernel and the guest of OS
VM has two levels (full and partial VM)
Containers take advantage of the c-groups and namespaces to isolate the jobs based on the resources.

The advantage of the Containers over the VM is that it requires fewer layers (OS and hypervisor) which means it can be faster while it could be a security threat.
Docker CLi → Docker daemon which produces the instant of the program
Docker daemon can pull it from the registry or we build it with CLI
Build image need a docker file () is a text file that docker is used to build an image of the docker

Dockerfile:
The idea for the Base image is to be as small as possible. It can have two strategies that to choose the image that is the closest to what we need already included or to use the only what is needed and then manually add whatever we need.
Docker images could have versions and they are documented as tags

Day 2 (04/12/2019) Summary

Task1:
We tried to build our own base image from an existing Dockerfile in the Horovod Repo that was designed for CPU-only usage. When using the correct python, tensorflow, keras and ubuntu version this image could be built. After resolving all dependency issues it worked to build the final container using only CPUs instead of GPUs. Running this container image worked both locally using Docker and on Piz Daint using Sarus.
We built the non-working image from yesterday using Sarus on Piz Daint where GPUs exist and this worked after we fixed all dependency issues.

Successfully run code(non_parallel version) in container by the following steps:

1.go to the source directory of local machine and run the following:
`$ docker build -t wfppdl/parallel_training:v1.0 -f ../docker/Dockerfile_parallel .`

2.Run for test
`$ docker run -ti wfppdl/parallel_training:v1.0`

3. Save the image to tar and transfer to the Daint supercomputer
`$ docker save -o wffppdl-training.tar wfppdl/parallel_training:v1.0`

4. copy to daint
`$ scp wffppdl-training.tar hck04@daint:/users/hck04`

5. log in the supercomputer :
`$ ssh hck04@daint`

6. transfer the tar file to daint and extract
`$ sarus load wffppdl-training.tar wffppdl/trainingV1.0`

7 Reserve the resource
`$ salloc -C gpu --reservation=esiwace_1 --time=00:20:00`

8 run the container

```
$ srun -C gpu  sarus run --mount=type=bind,source=/users/hck04/splits,destination=/splits
load/wffppdl/trainingV1.0


9 Export the files to the host

srun -C gpu  sarus run --mount=type=bind,source=/users/hck04/splits,destination=/splits --
mount=type=bind,source=/users/hck04/results/model_data_keras2,destination=/src/model
_data_keras2 load/wffppdl/trainingV1.0
```

Task2:
We have managed to install the sarus locally and we have checked the ability to pull from the docker hub. We have created an account in the docker hub and push the docker image that we made yesterday and we want to pull it directly to sarus and run it in the sarus.

Sample data is uploaded to the Daint to have the data to be mounted into the container for testing the synching process.

We faced the issue that the stager logger is logging inside the container and therefore, it is gone after the execution of the container. The source code was adopted to generate the log file as standard output (stdout) as well which can be accessed after the container is finished.

Day 3 (05/12/2019) Summary

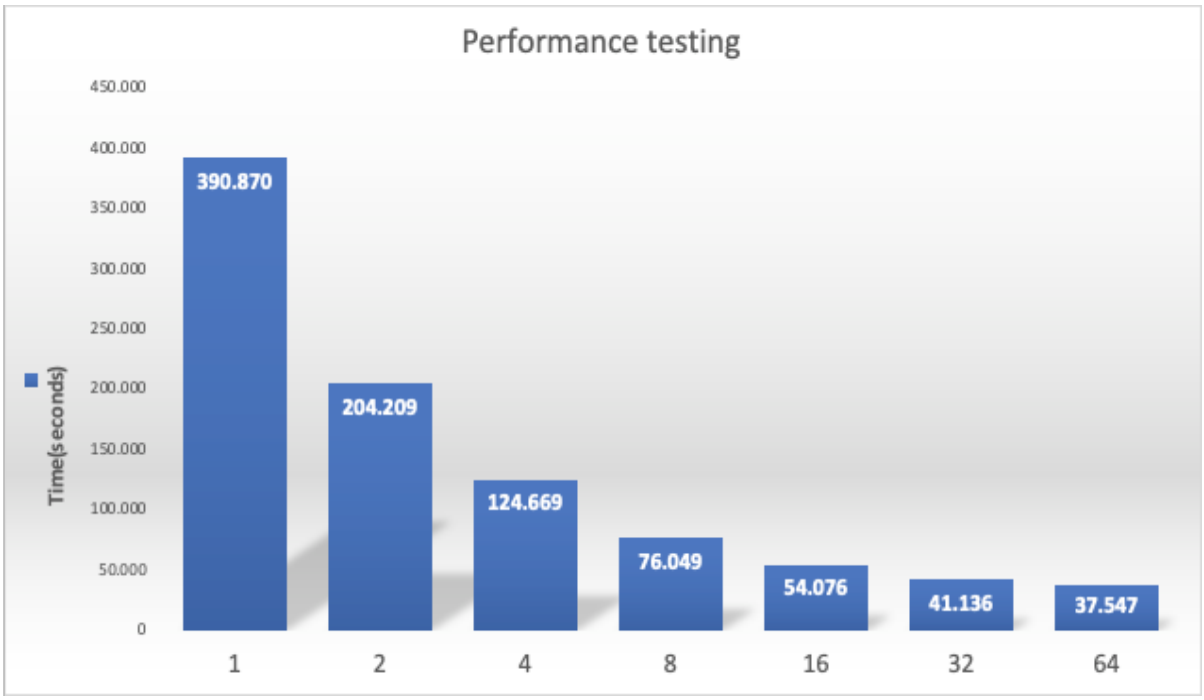
Task1:
Check open MPI version inside container
#docker run --entrypoint=bash -ti wfppdl/parallel_training:v2.0
Day 2 we tried to containerize the non-parallel training part “kitti_train.py”, it is successful. The main task today is to use the parallel training version “kitti_train_horovod.py” which is based on horovod and keras framework for containerzing.

The issue we faced is the openMPI version is not compatible with the system. We tried two strategies to address it a) Replace/downgrade the openMPI version from 4.0 to 3.1.4 in base image, and b) downgrade openMPI in our image “dockerfile_parallel_v2.1”
In the end approach a) worked and the container can now run with an arbitrary number of GPUs (one GPU per Node) in parallel.
The container can be found here:
https://hub.docker.com/repository/docker/janvog/parallel_training

In this DL training process, we use 15 number epochs, 5090 training samples, 8 images/frames for each sample, with 128*160*3 for dimensions for each frame. 15 batch size and 500 per epoch for each training (one GPU). We test the training performance by varying the GPUs numbers, and the results are demonstrated below table and image

Performance testing

Num of GPUs	(day, sec,microseconds)	Time (Seconds)
1	(0,390,870491)	390.870
2	(0,204,208980)	204.209
4	(0,124,669368)	124.669
8	(0, 76, 48956)	76.049
16	(0,54,76208)	54.076
32	(0,41,135555)	41.136
64	(0,37,546803)	37.547



Task2:
The Task 2 docker image is built again with base image of ubuntu and we were able to build all the dependencies in the new image and managed to run it local (it was pushed to Github). Then image was uploaded to the Daint and we were able to run it, but we had an issue with “get.size function” to identify the number of processors to be used for load balancer. We managed to address the issues with the code and run the Pystager inthe container, even though a problem in the configuration file prevented us to run the code until the end. But in general, we were successfully port the code and all the dependencies and managed to run it on the Dain through Sarus.

Shared Notes:

Final Conclusion and comments

- Short feedback about your experiences
- Obstacles you encountered, and how you solved them
- Lessons that you would like to share with other teams, e.g., suggestions on how to improve the process, better documentation, etc
- Last but not least, any general comments about this Container Hackathon for Modellers will be really useful for the organisers.
- We are interested in continuing the project on developing on Containerized workflow on CSCS resources and we will apply for preparatory project on.

Useful Links

Dockerfile reference:
https://docs.docker.com/engine/reference/builder/
Docker run reference:
https://docs.docker.com/engine/reference/run/
Horovod example of Dockerfile:
https://hub.docker.com/r/horovod/horovod/tags?page=1&name=0.16.2
Horovod examples:
https://sarus.readthedocs.io/en/latest/cookbook/tensorflow_horovod/tf_hvd.html
https://github.com/horovod/horovod/blob/master/Dockerfile.test.cpu

Some extra links and tricks:
Vagrants to make the environment built easy
https://www.vagrantup.com/
Docker vs. VM :
https://geekflare.com/docker-vs-virtual-machine/
“sudo !!” Is running the previous cmd with sudo privileges

Sarus in Daint reference:
https://user.cscs.ch/tools/containers/sarus/