LUMI

Starting with Machine Learning Frameworks on LUMI

Case Study with the Megatron-DeepSpeed

Goals

This presentation shows how to use a pre-trained GPT-3 model with the Megatron-DeepSpeed framework and AMD GPU architecture on LUMI

It shows how to:

- Combine basic ML workflow with HPC environment
- Use base containers with popular ML frameworks (torch and friends)
- Extend base container with custom modules (DeepSpeed, Megatron)
- Run custom container on a GPU compute node

What is Megatron-DeepSpeed?

- It is Microsoft's developed integration of their DeepSpeed library and NVIDIA's Megatron-LM framework
- DeepSpeed is an open source deep learning optimization library for PyTorch
- Megatron-LM is a framework for training large transformer language models at scale

What is LUMI Software Stack

- LUMI provides software stack that combines:
  - Programing Environment (compilers, libraries, networking stack)
  - ROCm runtime and development environment for AMD GPUs
  - Software libraries, tools and application codes
  - Base container images for common frameworks

Software requirements

- Common software environment is build on top of:
  - ROCm (AMD GPUs specific)
  - Python
  - PyTorch

How to start with ML on LUMI?

- Containers are the way for ML frameworks on LUMI
- Base images with most popular elements are provided
  - Enable LUMI specific inter-node communication
  - Enable recent ROCm runtime for AMD GPUs
  - Provide more recent environments than system's native
  - Ease extending with proper requirements included
- Public images may underperform or fail

Using LUMI base images

- LUMI uses Singularity containers
- As a part of the Software Stack
  - Image access via Modules environment (common HPC tool)
  - Module generation via EasyBuild recipe (another HPC tool)
- Natively using sif image file directly
- Run images on compute nodes
  - Remote allocation and execution with Slurm (popular HPC workload manager)

Which base images are available ?

- PyTorch (2.0.1, 2.1.0, 2.2.0)
- Tensorflow (2.11.1 + Horovod)
- Jax (0.4)
- Alphafold
- ROCm (5.4.x, 5.5.x, 5.6.x)
- MPI4Py

- See also https://lumi-supercomputer.github.io/LUMI-EasyBuild-docs/ (LUMI Software Library) entries with a singularity container badge

Executing a simple container

- Say **`$SIF`** is the MPI4Py container image file
- Base images use Conda environments
- Simple wrapper script helps with Conda initialization

```
#!/bin/bash -e

$WITH_CONDA

python3 "$@"
```

- Run the container

```
singularity exec $SIF ./conda_wrapper.sh -c "import mpi4py"
```

  - runs locally (on a login node)
  - executes single python import command

Executing container on a compute node

- Prepend singularity exec command with **srun** launcher
- Required options are:
  - project account: **`$SLURM_ACCOUNT, --account`**
  - partition: **`$SLURM_PARTITION, --partition`**
- Other practical options:
  - processes per node: --ntasks-per-node
  - gpu devices per node: --gpu-per-node
  - --time
- Run the container on a remote node

```
srun singularity exec $SIF ./conda_wrapper.sh -c "import mpi4py"
```

- Slurm job script for batch submissions

Running container on a compute node

- Run real workload within the container

```
srun -n 2 singularity exec $SIF ./conda_wrapper.sh ./mpi_workload.py
```

  - fails with ImportError: cannot open shared object file
  - missing some libraries from the host side
- Final command with host bindings

```
srun -n 2 singularity exec
-B \
/var/spool/slurmd,/opt/cray, \
/usr/lib64/libcxi.so.1,/usr/lib64/libjansson.so.4 \
$SIF ./conda_wrapper.sh ./mpi_workload.py
```

- Could it be any easier?

Running container with the Software Stack

- LUMI Software Stack is built on the Cray Programming Environment
- It uses environmental modules
- Initialize with
  - module load LUMI
- Select target architecture (with container being specific case)
  - module load partition/container
- Enable EasyBuild framework (HPC software management tool)
  - module load EasyBuild-user
- Generate modules and environment
  - eb mpi4py-3.1.4-rocm-5.4.5-python-3.10-singularity-20231110.eb
- Run the container

```
module load LUMI partition/container
mpi4py/3.1.4-rocm-5.4.5-python-3.10-singularity-20231110

srun -n 2 singularity exec $SIF ./conda_wrapper.sh ./mpi_workload.py
```

Extending base container

- PyToch container will serve the requirements, i.e.:
  - rocm-5.6.1
  - python-3.10
  - pytorch-v2.2.0
- Use virtual environment from within the container and install additional packages in the working directory (filesystem)
  - This is not recommended with large file counts
  - Practical for development/testing
- Create a Singularity overlay with custom packages
  - Filesystem friendy
  - Encapsulates environment within a single flat file

Using venv in the Conda container

- Local (login node) session is enough to install packages
- Run the PyTorch container
- Activate Conda
- Initialize virtual environment
- Activate the environment

```
$WITH_CONDA
python3 -m venv --system-site-packages ml-framework-env
source ml-framework-env/bin/activate
```

Adding DeepSpeed module

```
git clone --recursive https://github.com/microsoft/DeepSpeed.git
cd DeepSpeed
pip install .[dev,1bit,autotuning]
```

Adding Megatron module

```
git clone https://github.com/microsoft/Megatron-DeepSpeed.git
cd Megatron-DeepSpeed
pip3 install pybind11 nltk transformers
```

Using Singularity overlay

- Create install script
  - requirements definition (DeepSpeed, Megatron-Deepspeed)
  - pip install command
- Install modules from within the container in scratch directory
- Encapsulate scratch directory in an overlay image file
- Embed an overlay image in the original sif or use --overlay option with singularity exec to bind overlay to the original image

Executing ML tasks on compute node

- Execute container on a compute node with srun interactively
- Allocate 8 GPUs
- Initialize Conda **`(pytorch)`**
- Activate virtual environment **`(ml-framework-env)`**

```
srun --pty --interactive --gpus-per-node=8 --exclusive \
singularity exec $SIF bash

Singularity> $WITH_CONDA

(pytorch) Singularity> source ./ml-framework-env/bin/activate

(ml-framework-env) (pytorch) Singularity> cd ./ml-framework-env
```

Prepare datasets

- Download dataset

```
cd ./Megatron-DeepSpeed/dataset

wget https://huggingface.co/bigscience/misc-test-data/resolve/main/stas/oscar-1GB.jsonl.xz

xz -d oscar-1GB.jsonl.xz

bash download_vocab.sh
```

- Pre-process data

```
export BASE_SRC_PATH=./Megatron-DeepSpeed

export BASE_DATA_PATH=${BASE_SRC_PATH}/dataset

python3 ${BASE_SRC_PATH}/tools/preprocess_data.py \
--input ${BASE_DATA_PATH}/oscar-1GB.jsonl --output-prefix ${BASE_DATA_PATH}/my-gpt2 \
--vocab-file ${BASE_DATA_PATH}/gpt2-vocab.json --dataset-impl mmap \
--tokenizer-type GPT2BPETokenizer --merge-file ${BASE_DATA_PATH}/gpt2-merges.txt \
--append-eod --workers 8
```

Using pre-trained model checkpoint

- Retrieve model checkpoint to test the framework
- Check pre-trained model with a text generation task
- Generate three samples based on prompt

Retrieve pre-trained model checkpoint

```
from transformers import GPT2LMHeadModel
from transformers import GPT2Tokenizer
from transformers import set_seed
import torch

torch_device = "cuda" if torch.cuda.is_available() else "cpu"

tokenizer =
GPT2Tokenizer(vocab_file='./Megatron-DeepSpeed/dataset/gpt2-vocab.json'
, merges_file='./Megatron-DeepSpeed/dataset/gpt2-merges.txt')

model1 =
GPT2LMHeadModel.from_pretrained('jiagaoxiang/gpt3-125M-8000iter',
pad_token_id=tokenizer.eos_token_id).to(torch_device)
```

Define a prompt

```
model_inputs = tokenizer('LUMI is the fastest supercomputer in Europe and
the fifth fastest globally', return_tensors='pt').to(torch_device)
```

Generate samples

```
# Set seed
set_seed(1)
# Set top_k = 50, top_p = 0.95, and num_return_sequences = 3
sample_outputs = model1.generate(
    **model_inputs,
    max_new_tokens=40,
    do_sample=True,
    top_k=50,
    top_p=0.95,
    num_return_sequences=3,
)
# Output samples
for i, sample_output in enumerate(sample_outputs):
  print("{}: {}".format(i, tokenizer.decode(sample_output, skip_special_tokens=True)))
```

Result 😆

- *LUMI is the fastest supercomputer in Europe and the fifth fastest globally by the United States.*

  There are a lot of problems with Linux since we cannot support the installation of Linux on an operating system.

  I guess that's what Windows NT support is all about.
- *LUMI is the fastest supercomputer in Europe and the fifth fastest globally. For more information and advice, read the article by E.L.C.B and more info.*

  If your hosting services are slow or cumbersome, you can start by going from one website
- *LUMI is the fastest supercomputer in Europe and the fifth fastest globally. The fastest in the world, the UK and the USA is second only to the United States.*

  The latest from the Crypt of the Year is the World Wide Web, which is an astonishingly

Next steps

- Pre-train model and tune parameters
- Distribute training among multiple GPU nodes (adding PyTorch-Lighting module for instance)

Conclusion

- You can run common ML frameworks on LUMI
- Look for AMD GPU enabled versions
- Base container images provide easy to start, performance friendly approach
- Distributed, multi-node processing require specific environment

Materials used

- LUMI Docs https://docs.lumi-supercomputer.eu/
- LUMI Software Library https://lumi-supercomputer.github.io/LUMI-EasyBuild-docs/
- AMD Blog https://blogs.rocm.amd.com/artificial-intelligence/megatron-deepspeed-pretrain/
- SingularityCE User Guide https://docs.sylabs.io/guides/3.11/user-guide/
- Conda User guide https://conda.io/projects/conda/en/latest/user-guide/
- Slurm Documentation https://slurm.schedmd.com/archive/slurm-22.05.10/

LUMI

Maciej Szpindler

m.szpindler@cyfronet.pl

Follow us

Twitter: @LUMIhpc

LinkedIn: LUMI supercomputer

YouTube: LUMI supercomputer

www.lumi-supercomputer.eu

contact@lumi-supercomputer.eu

EuroHPC Joint Undertaking

The acquisition and operation of the EuroHPC supercomputer is funded jointly by the EuroHPC Joint Undertaking, through the European Union's Connecting Europe Facility and the Horizon 2020 research and innovation programme, as well as the of Participating States FI, BE, CH, CZ, DK, EE, IS, NO, PL, SE.

Leverage from the EU 2014–2020

European Union
European Regional Development Fund

REGIONAL COUNCIL OF KAINUU

EURO