CSC
Training for Brilliant Minds

Machine Learning on Puhti

Part 2: Scaling up and using resources efficiently

June 10, 2020
Mats Sjöberg – mats.sjoberg@csc.fi

Overview

Data storage

GPU utilization

Multi-GPU and multi-node jobs

Singularity containers

Data storage

Data storage on Puhti

- Disk space and *number of files* are limited on Puhti!
  $\rightarrow$ We want to ensure that the shared (Lustre) filesystem works efficiently for everyone!
- Useful command: `csc-workspaces`

	Owner	Path	Capacity	Number of files	Cleaning
home	Personal	`/users/<user-name>`	10 GiB	100 000 files	No
projappl	Project	`/projappl/<project>`	50 GiB	100 000 files	No
scratch	Project	`/scratch/<project>`	1 TiB	1 000 000 files	Yes - 90 days

Data quotas can be increased via MyCSC!

https://docs.csc.fi/computing/disk/

Using Allas

- store big datasets in Allas, CSC's object storage
- download them to project scratch prior to computation
- you can also upload trained models (or keep in projappl)

```
$ module load allas
$ allas-conf
$ cd /scratch/<your-project>
$ swift download <bucket-name> your-dataset.tar
```

Large number of files

- Many datasets contain a large number of small files
- Shared filesystem (Lustre) performs poorly in this scenario
  $\rightarrow$ noticable slowdowns for all Puhti users!

Consider alternatives:

- packaging your dataset into larger files
- use NVME fast local storage on GPU nodes

Using more efficient data formats

Instead of many small files, use one or a few bigger files.

Examples:

- TensorFlow's TFRecord format
- HDF5
- LMDB
- ZIP, for example via Python's `zipfile` library

Fast local NVME drive

- All GPU nodes have a local NVME drive
- Just add `nvme:<number-of-GB>` to sbatch `--gres` flag

```
#!/bin/bash
#SBATCH --account=<project>
#SBATCH --partition=gpu
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=10
#SBATCH --mem=64G
#SBATCH --time=1:00:00
#SBATCH --gres=gpu:v100:1,nvme:100

tar xf /scratch/<your-project>/your-dataset.tar -C $LOCAL_SCRATCH

srun python3 myprog.py --data_dir=$LOCAL_SCRATCH <options>
```

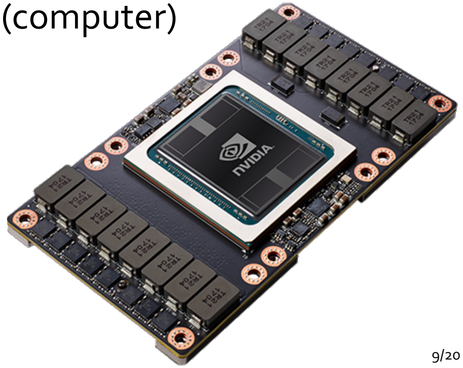
GPU utilization

GPU utilization

GPUs are an expensive resource compared to CPUs ($\times$ 60 BUs!)
$\rightarrow$ GPU should be maximally utilized!

For a running job:

- use `squeue` to find out on what node (computer) it is running
- ssh into that node, e.g., `ssh r01g01`
- find the process id of your job with `ps -u $USER`
- run `nvidia-smi`



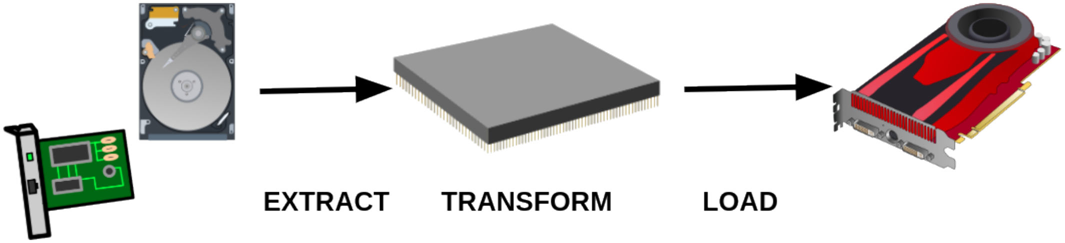
GPU utilization

For a finished job:

- run `gpuseff <jobid>`
- shows GPU utilisation statistics for the whole running time
- note: gpuseff is currently in testing usage, and still under development

You can always contact our service desk if you need advice on how to improve your GPU utilization!

Using multiple CPUs for ETL



- Common bottle-neck: CPU cannot keep up with GPU
- GPU has to wait for more data ...
- Solution: use more CPUs (they are “cheaper” than GPUs!)

A good rule of thumb in Puhti is to reserve 10 CPUs per GPU (as there are 4 GPUs and 40 CPUs per node).

Using multiple CPUs for ETL

In Slurm scripts:

```
#SBATCH --cpus-per-task=10
```

Note: using multiple CPUs *not* automatic, code needs to support it!

For example in TensorFlow:

```
dataset = dataset.map(..., num_parallel_calls=10)
dataset = dataset.prefetch(buffer_size)
```

PyTorch:

```
train_loader = torch.utils.data.DataLoader(..., num_workers=10)
```

Multi-GPU and multi-node jobs

Multi-GPU

Many frameworks support multi-GPU within a single node.

Slurm script:

```
#SBATCH --gres=gpu:v100:4
```

TensorFlow:

```
mirrored_strategy = \
  tf.distribute.MirroredStrategy()
with mirrored_strategy.scope():
    model = Sequential(...)
    model.add(...)
    model.add(...)
    model.compile(...)
```

PyTorch:

```
model = MyModel(...)
if torch.cuda.device_count() > 1:
    model = nn.DataParallel(model)
model.to(device)
```

Multi-GPU and multi-node

- A single node has 4 GPUs
- If you need more than 4 GPUs, we recommend Horovod
- Supported for TensorFlow and PyTorch on Puhti
- Uses MPI and NCCL for interprocess communication
- Modules with `-hvd suffix`

Try:

```
module avail hvd
module load tensorflow/2.0.0-hvd
```

Slurm example for Horovod

Example slurm script that uses 8 GPUs across two computers

- MPI terminology: 8 tasks, 2 nodes
- Each task is 1 GPU and 10 CPUs

```
#!/bin/bash
#SBATCH --account=<project>
#SBATCH --partition=gpu
#SBATCH --ntasks=8
#SBATCH --nodes=2
#SBATCH --cpus-per-task=10
#SBATCH --mem=32G
#SBATCH --time=1:00:00
#SBATCH --gres=gpu:v100:4

srun python3 myprog.py <options>
```

Singularity containers

Singularity on Puhti

- Puhti supports Singularity-based containers
- Some of our modules use it – just remember to prefix commands with `singularity_wrapper exec`
- You can also convert your own Docker containers, see: https://docs.csc.fi/computing/containers/run-existing/

Conversion (preferrably not in login nodes!):

```
$ singularity build pytorch_20.03-py3.sif \
                docker://nvcr.io/nvidia/pytorch:20.03-py3
```

Extract from Slurm script:

```
srun singularity_wrapper exec --nv pytorch_20.03-py3.sif \
    python3 myprog <options>
```

Special Singularity-based applications

Specialized Singularity-based applications not shown with default modules

Example: Turku neural parser

```
$ module use /appl/soft/ai/singularity/modulefiles/
$ module load turku-neural-parser/fi-en-sv-gpu
$ echo "Minulla on koira." | singularity_wrapper run \
    stream fi_tdt parse_plaintext
```

Thank you!