Build PyTorch on HPC Cluster

Bosung Lee

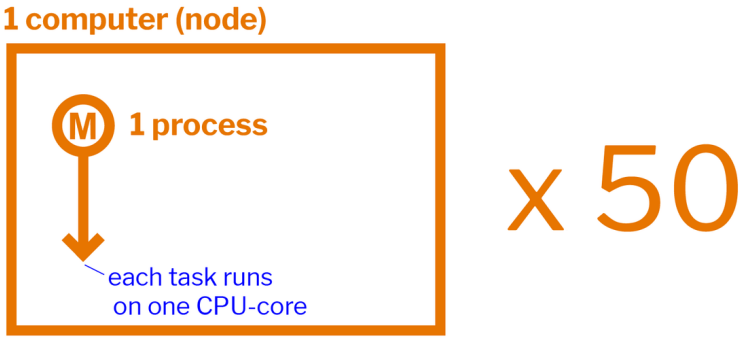
bslee@nextfoam.co.kr

CPO, NEXTFOAM

Four Basic Types of Parallel Programming

Embarassingly Parallel

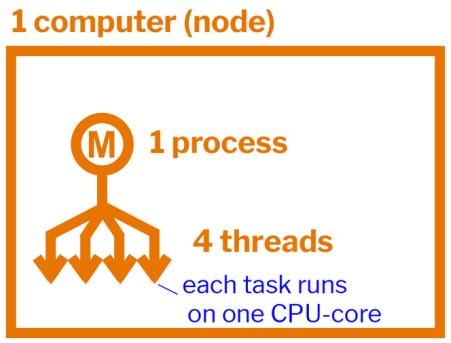
- each task can be run completely independently of other tasks
- no communication required between tasks



x 50

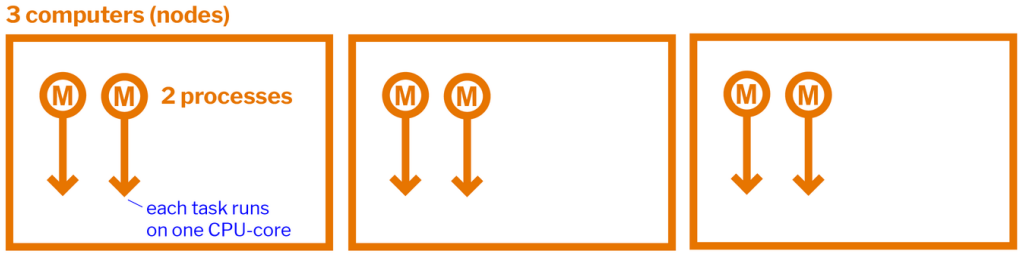
Shared-Memory Parallel

- tasks are run as threads on separate CPU-cores
- light level of communication between the cores working on each task



Distributed Memory Parallel

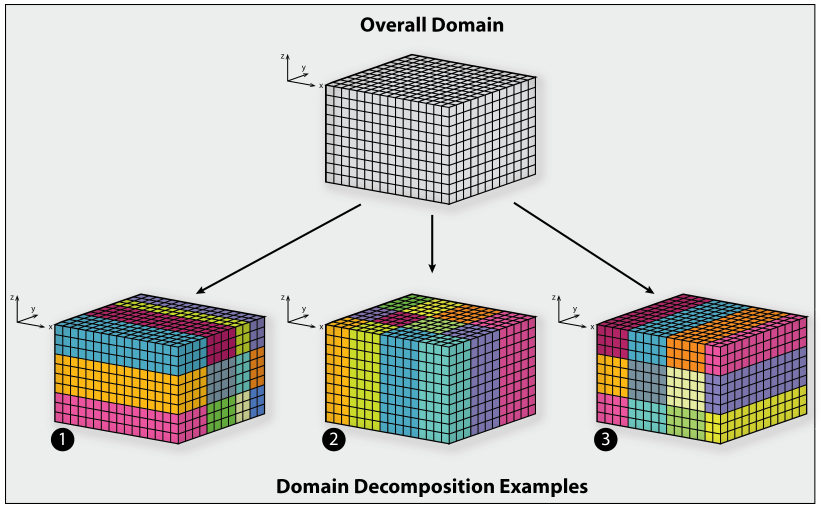
- multiple processes do not share the same space in memory
- high level of communication between different tasks



Accelerator Parallel

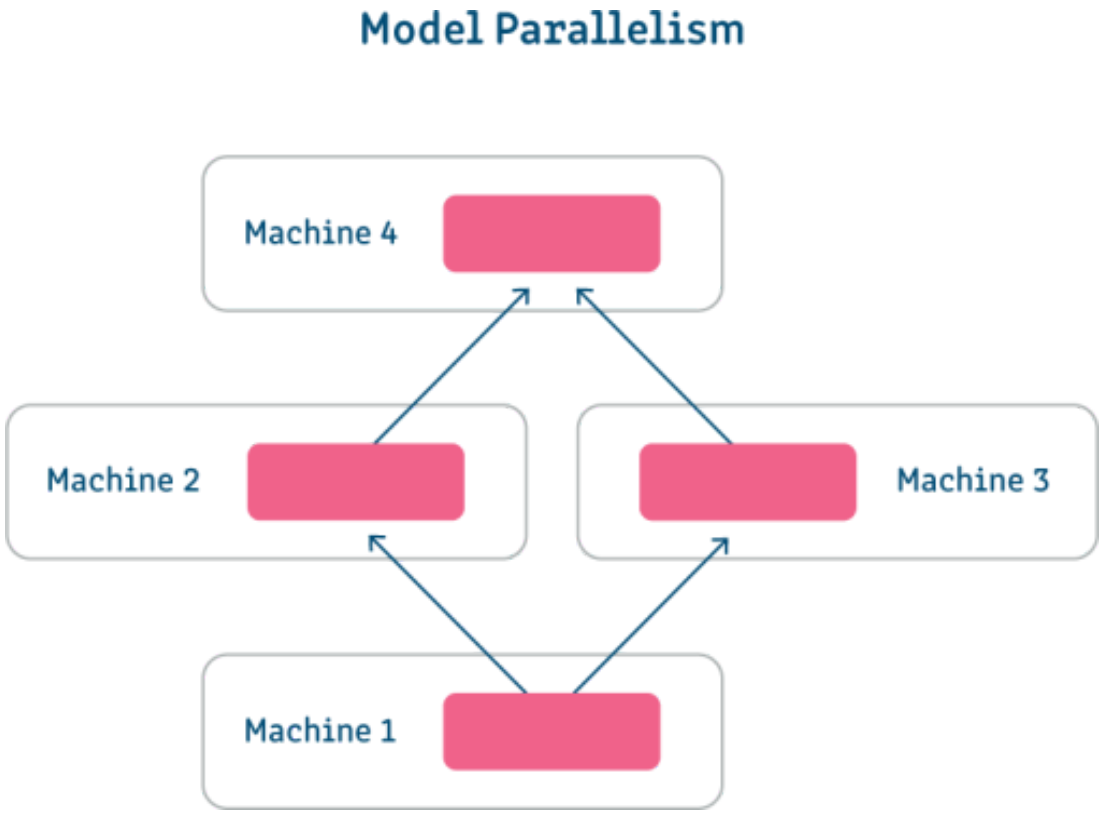
- uses different types of computer hardware - such as GPU and FPGA

Domain Decomposition

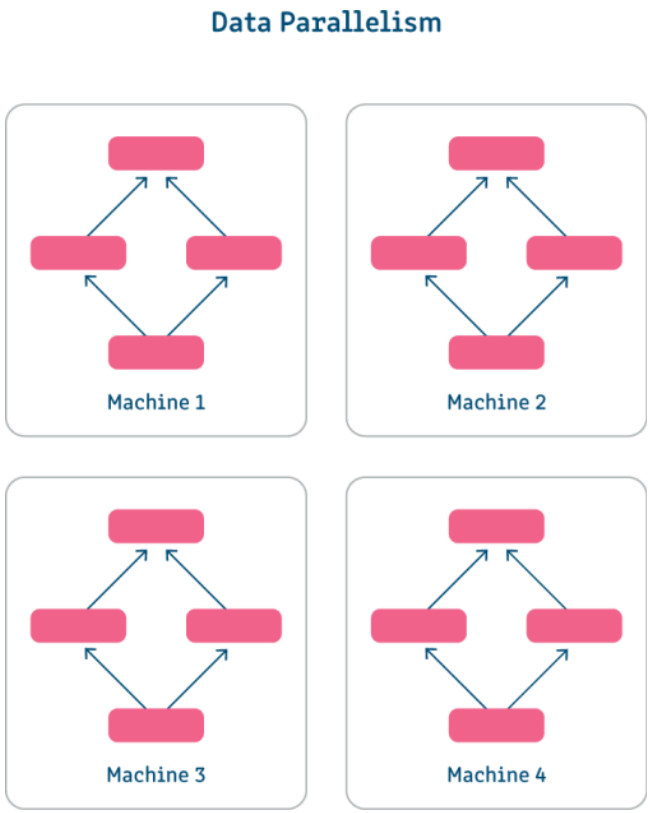


Distributed Data Parallel vs Model Parallel

- the model itself is divided into parts that are trained simultaneously across different worker nodes



- Each worker node trains a copy of the model on a different batch of training data



Build HPC for PyTorch Distributed Data Parallel

1. HPC Cluster Setup
    - Configure communication backends (gloo, mpi, nccl)
2. Install miniconda for all compute nodes
3. Install Pytorch with MPI Backend
4. Run distributed training using MPI backend

Communication Backends

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✗	✓	?	✗	✓
recv	✓	✗	✓	?	✗	✓
broadcast	✓	✓	✓	?	✗	✓
all_reduce	✓	✓	✓	?	✗	✓
reduce	✓	✗	✓	?	✗	✓
all_gather	✓	✗	✓	?	✗	✓
gather	✓	✗	✓	?	✗	✓
scatter	✓	✗	✓	?	✗	✓
reduce_scatter	✗	✗	✗	✗	✗	✓
all_to_all	✗	✗	✓	?	✗	✓
barrier	✓	✗	✓	?	✗	✓

Which backend to use?

- Rule of thumb
  - Distributed GPU training : NCCL
  - Distributed CPU training. : Gloo
- GPU hosts with InfiniBand : NCCL
- GPU hosts with Ethernet : NCCL
- CPU hosts with InfiniBand : Gloo, MPI
- CPU hosts with Ethernet : Gloo, MPI

Install OpenMPI 4.1.6

Login to the headnode as root and install required packages to compile. Before installing the OpenMPI from source, remove default openmpi packages

```
root@hostname:~# apt remove openmpi*
root@headnode:~# apt-get -y update
root@headnode:~# apt-get -y install build-essential flex zlib1g-dev libgmp-dev libmpfr-dev
```

Download `openmpi 4.1.6` source and install to `/opt/openmpi-4.1.6` directory

```
root@headnode:~# wget https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-4.1.6.tar.gz
root@headnode:~# tar zxf openmpi-4.1.6.tar.gz ; rm openmpi-4.1.6.tar.gz
root@headnode:~# cd openmpi-4.1.6
root@headnode:~# ./configure --prefix=/opt/openmpi-4.1.6
root@headnode:~# make -j 4 all
root@headnode:~# make install
root@headnode:~# echo 'export PATH=$PATH:/opt/openmpi-4.1.6/bin' >> /etc/bash.bashrc
```

For GPUs, ***--with-cuda*** or ***--with-rocm*** option is needed at `configure` step

Install miniconda for all compute nodes

Login to the headnode as root and install miniconda for all compute nodes

```
root@headnode:~# wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
-O /opt/miniconda-installer.sh
root@headnode:~# bash /opt/miniconda-installer.sh
```

Once the installation is finished, type `[yes]` to initialize Miniconda

```
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
[no] >>> yes
```

Set the `auto_activate_base` to `false` not to activate the base environment at startup

```
root@headnode:~# conda config --set auto_activate_base false
```

Add the conda initialization script at `/etc/bash.bashrc` for all users

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/opt/miniconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/opt/miniconda3/etc/profile.d/conda.sh" ]; then
        . "/opt/miniconda3/etc/profile.d/conda.sh"
    else
        export PATH="/opt/miniconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

Copy miniconda installation to all compute nodes

```
root@headnode:~# conda update -y --all
root@headnode:~# bpushdir /opt/miniconda3 /opt
root@headnode:~# bpush /etc/bash.bashrc /etc
```

Build Pytorch with MPI backend

To get MPI backend for `torch distributed` working you need to recompile PyTorch. To compile Pytorch with MPI, login as a user and create a virtual environment

```
bosung@node001:~$ conda create -n torch-mpi python=3.8
bosung@node001:~$ conda activate torch-mpi
```

Download pytorch and install required packages, and compile

```
(torch-mpi) bosung@node001:~$ git clone https://github.com/pytorch/pytorch
(torch-mpi) bosung@node001:~$ pip install pyyaml typing_extensions torchvision
(torch-mpi) bosung@node001:~$ conda install -c conda-forge libstdcxx-ng=12
(torch-mpi) bosung@node001:~$ cd pytorch
(torch-mpi) bosung@node001:~/pytorch$ python3 setup.py develop --user
```

Run distributed training using MPI backend

Login to a head node and activate the virtual environment. Then clone the repo and download dataset to train a simple CNN on the MNIST data set.

```
bosung@node001:~$ conda activate torch-mpi
(torch-mpi) bosung@node001:~$ git clone https://github.com/PrincetonUniversity/install_pytorch.git
(torch-mpi) bosung@node001:~$ cd install_pytorch
(torch-mpi) bosung@node001:~/install_pytorch$ python download_mnist.py
```

To print the stdout, install `sys` and modify the source code

```
print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
      epoch, batch_idx * len(data), len(train_loader.dataset),
      100. * batch_idx / len(train_loader), loss.item()))
sys.stdout.flush()
```

Run the DDP training multi cores.

```
(torch-mpi) bosung@node001:~/install_pytorch$ mpirun -np 16 python mnist_classify.py --epochs=3
```