PyTorch

blocked URL

Opis

PyTorch je python knjižnica namijenjena razvoju aplikacija temeljenih na dubokom učenju koja se oslanja na ubrzanje grafičkim procesorima. Glavne prednosti koje PyTorch knjižnica pruža su imperativni pristup programiranju na "python" način, kroz sučelje koje omogućuje lakše otkrivanje grešaka i koje je prilagođeno postojećim python znanstvenim knjižnicama.

Verzije

verzija	modul	python	Supek	Padobran
1.8.0	scientific/pytorch/1.8.0-ngc	3.8	✅	
1.14.0	scientific/pytorch/1.14.0-ngc	3.8	✅	
2.0.0	scientific/pytorch/2.0.0	3.10	✅	✅
	scientific/pytorch/2.0.0-ngc	3.10	✅	

Korištenje aplikacije na Supeku

Python aplikacije i knjižnice na Supeku su dostavljene u obliku kontejnera i zahtijevaju korištenje wrappera kao što je opisano ispod.

Više informacija o python aplikacijama i kontejnerima na Supeku možete dobiti na sljedećim poveznicama:

- Python, pip i conda
- Apptainer

Dokumentacija

- Službena stranica - https://pytorch.org/
- Priručnik - https://pytorch.org/docs/stable/index.html
- distributed - https://pytorch.org/docs/stable/distributed.html
- torchrun
    - API - https://pytorch.org/docs/stable/elastic/run.html
    - primjeri - https://pytorch.org/tutorials/intermediate/dist_tuto.html
- accelerate
    - API - https://huggingface.co/docs/accelerate/index
    - primjeri - https://huggingface.co/docs/accelerate/usage_guides/training_zoo

Supek

Ispod se nalaze primjeri pozivanja naredbi i aplikacija unutar kontejnera i aplikacija umjetnog benchmarka koji testira performanse na modelu Resnet50.

Pozivanje naredbi unutar kontejnera

test.sh

```
    [korisnik@x3000c0s25b0n0] $ module load scientific/pytorch/1.14.0-ngc
[korisnik@x3000c0s25b0n0] $ run-command.sh pip3 list
INFO:    underlay of /etc/localtime required more than 50 (95) bind mounts
INFO:    underlay of /usr/bin/nvidia-smi required more than 50 (474) bind mounts
13:4: not a valid test operator: (
13:4: not a valid test operator: 510.47.03
Package                Version
---------------------- ------------------------------
absl-py                1.3.0
accelerate             0.19.0
apex                   0.1
appdirs                1.4.4
argon2-cffi            21.3.0
argon2-cffi-bindings   21.2.0
asttokens              2.2.1
...
```

Izvršavanje PyTorch koda na jednom grafičkom procesoru

singlegpu.py

```
    # source
# - https://github.com/horovod/horovod/blob/master/examples/pytorch/pytorch_synthetic_benchmark.py

import os
import time

import torch
import torch.nn as nn
import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision.models import resnet50
from torchvision.datasets import FakeData
from torchvision.transforms import ToTensor

def main():

    # vars
    batch = 256
    samples = 256*100
    epochs = 1

    # model
    model = resnet50(weights=None)
    model.cuda()
    optimizer = optim.SGD(model.parameters(), lr=0.001)
    loss_fn = nn.CrossEntropyLoss()

    # data
    dataset = FakeData(samples,
                       num_classes=1000,
                       transform=ToTensor())
    loader = DataLoader(dataset,
                        batch_size=batch,
                        shuffle=False,
                        num_workers=1,
                        pin_memory=True)

    # train
    for epoch in range(epochs):
        start = time.time()
        for batch, (images, labels) in enumerate(loader):
            images = images.cuda()
            labels = labels.cuda()
            outputs = model(images)
            classes = torch.argmax(outputs, dim=1)
            loss = loss_fn(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            if (batch%10 == 0):
                print('--- Epoch %i, Batch %3i / %3i, Loss = %0.2f ---' % (epoch,
                                                                            batch,
                                                                            len(loader),
                                                                            loss.item()))
        elapsed = time.time()-start
        imgsec = samples/elapsed
        print('--- Epoch %i finished: %0.2f img/sec ---' % (epoch,
                                                            imgsec))

if __name__ == "__main__":
    main()
```

singlegpu.sh

```
    #!/bin/bash

#PBS -q gpu
#PBS -l ngpus=1

# pozovi modul
module load scientific/pytorch/2.0.0-ngc

# pomakni se u direktorij gdje se nalazi skripta
cd ${PBS_O_WORKDIR:-""}

# potjeraj skriptu korištenjem run-singlegpu.sh
run-singlegpu.sh singlegpu.py
```

torchrun/distributed

Torchrun & distributed

Korištenje wrappera `torchun-*.sh` ili `distributed-*.sh` je zamjenjivo u slučaju da je pytorch kod distribuiran `torch.distributed` modulom.

Aplikacija na više grafičkih procesora i jednom čvoru

multigpu-singlenode.py

```
    # source
# - https://pytorch.org/tutorials/intermediate/dist_tuto.html
# - https://pytorch.org/vision/main/generated/torchvision.datasets.FakeData.html
# - https://tuni-itc.github.io/wiki/Technical-Notes/Distributed_dataparallel_pytorch/#setting-up-the-same-model-
with-distributeddataparallel

import time

import torch
import torch.nn as nn
import torch.optim as optim
import torch.distributed as dist

from torch.utils.data import DataLoader
from torch.utils.data.distributed import DistributedSampler
from torch.nn.parallel import DistributedDataParallel as DDP

from torchvision.models import resnet50
from torchvision.datasets import FakeData
from torchvision.transforms import ToTensor

def main():

    # vars
    batch = 256
    samples = 25600
    epochs = 3

    # init
    dist.init_process_group("nccl")
    rank = dist.get_rank()
    ngpus = torch.cuda.device_count()

    # model
    model = resnet50(weights=None)
    model = model.to(rank)
    model = DDP(model, device_ids=[rank])
```

```
    optimizer = optim.SGD(model.parameters(), lr=0.001)
    loss_fn = nn.CrossEntropyLoss()

    # data
    dataset = FakeData(samples,
                       num_classes=1000,
                       transform=ToTensor())
    sampler = DistributedSampler(dataset)
    loader = DataLoader(dataset,
                        batch_size=batch//ngpus,
                        sampler=sampler,
                        shuffle=False,
                        num_workers=2,
                        pin_memory=True,)

    # train
    for epoch in range(epochs):
        start = time.time()
        for batch, (images, labels) in enumerate(loader):
            images = images.to(rank)
            labels = labels.to(rank)
            outputs = model(images)
            classes = torch.argmax(outputs, dim=1)
            loss = loss_fn(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            if (rank == 0) and (batch%10 == 0):
                print('epoch: %3d, batch: %3d, loss: %0.4f' % (epoch+1,
                                                               batch,
                                                               loss.item()))
        if (rank == 0):
            elapsed = time.time()-start
            img_sec = samples/elapsed
            print('Epoch complete in %s seconds [%f img/sec] ' % (elapsed, img_sec))

    # clean
    dist.destroy_process_group()

if __name__ == "__main__":
    main()
```

multigpu-singlenode.sh

```
    #!/bin/bash

#PBS -q gpu
#PBS -l ngpus=4
#PBS -l ncpus=16

# pozovi modul
module load scientific/pytorch/1.14.0-ngc

# pomakni se u direktorij gdje se nalazi skripta
cd ${PBS_O_WORKDIR:-""}

# potjeraj skriptu korištenjem torchrun-singlenode.sh
torchrun-singlenode.sh multigpu-singlenode.py
```

Aplikacija na više grafičkih procesora i više čvorova

multigpu-multinode.py

```
    # source
# - https://pytorch.org/tutorials/intermediate/dist_tuto.html
# - https://pytorch.org/vision/main/generated/torchvision.datasets.FakeData.html
```

```
# - https://tuni-itc.github.io/wiki/Technical-Notes/Distributed_dataparallel_pytorch/#setting-up-the-same-model-with-distributeddataparallel

import os
import time

import torch
import torch.nn as nn
import torch.optim as optim
import torch.distributed as dist

from torch.utils.data import DataLoader
from torch.utils.data.distributed import DistributedSampler
from torch.nn.parallel import DistributedDataParallel as DDP

from torchvision.models import resnet50
from torchvision.datasets import FakeData
from torchvision.transforms import ToTensor

def main():

    # vars
    batch = 256
    samples = 256*100
    epochs = 3

    # init
    dist.init_process_group("nccl")
    rank = int(os.environ['LOCAL_RANK'])
    global_rank = int(os.environ['RANK'])

    # model
    model = resnet50(weights=None)
    model = model.to(rank)
    model = DDP(model, device_ids=[rank])
    optimizer = optim.SGD(model.parameters(), lr=0.001)
    loss_fn = nn.CrossEntropyLoss()

    # data
    dataset = FakeData(samples,
                       num_classes=1000,
                       transform=ToTensor())
    sampler = DistributedSampler(dataset)
    loader = DataLoader(dataset,
                        batch_size=batch,
                        sampler=sampler,
                        shuffle=False,
                        num_workers=1,
                        pin_memory=True,)

    # train
    for epoch in range(epochs):
        start = time.time()
        for batch, (images, labels) in enumerate(loader):
            images = images.to(rank)
            labels = labels.to(rank)
            outputs = model(images)
            classes = torch.argmax(outputs, dim=1)
            loss = loss_fn(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            if (global_rank == 0) and (batch%10 == 0):
                print('epoch: %3d, batch: %3d/%3d, loss: %0.4f' % (epoch+1,
                                                                   batch,
                                                                   len(loader),
                                                                   loss.item()))
        if (global_rank == 0):
            elapsed = time.time()-start
            img_sec = samples/elapsed
            print('Epoch complete in %0.2f seconds [%0.2f img/sec] ' % (elapsed, img_sec))
```

```
    # clean
    dist.destroy_process_group()

if __name__ == "__main__":
    main()
```

multigpu-multinode.sh

```
    #!/bin/bash

#PBS -q gpu
#PBS -l select=8:ngpus=1:ncpus=4

# pozovi module
module load scientific/pytorch/1.14.0-ngc

# pomakni se u direktorij gdje se nalazi skripta
cd ${PBS_O_WORKDIR:-""}

# potjeraj skriptu korištenjem torchrun-multinode.sh
torchrun-multinode.sh multigpu-multinode.py
```

accelerate

Aplikacija na jednom čvoru

accelerate-singlenode-run.sh

```
    #!/bin/bash

#PBS -q gpu
#PBS -l select=1:ngpus=2:ncpus=8

# env
module load scientific/pytorch/2.0.0

# cd
cd ${PBS_O_WORKDIR:-""}

# run
accelerate-singlenode.sh accelerate-singlenode.py
```

accelerate-singlenode.py

```
    # source
# - https://github.com/horovod/horovod/blob/master/examples/pytorch/pytorch_synthetic_benchmark.py

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from accelerate import Accelerator

from torchvision import models
from torch.utils.data import DataLoader
from torchvision.datasets import FakeData
from torchvision.transforms import ToTensor

import os
import sys
import time
```

```
import pprint
import numpy as np

def main():

    # settings
    epochs = 3
    batch_size = 256
    image_number = 256*30
    model = 'resnet50'

    # accelerator
    accelerator = Accelerator()

    # model
    model = getattr(models, model)()
    model.to(accelerator.device)

    # optimizer
    optimizer = optim.SGD(model.parameters(), lr=0.01)
    loss_function = nn.CrossEntropyLoss()

    # loader
    data = FakeData(image_number,
                    num_classes=1000,
                    transform=ToTensor())

    loader = DataLoader(data,
                        batch_size=batch_size)

    # scheduler
    scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9)

    # prepare
    model, optimizer, loader, scheduler = accelerator.prepare(model,
                                                              optimizer,
                                                              loader,
                                                              scheduler)

    # fit
    for epoch in range(epochs):
        start = time.time()
        for batch, (images, labels) in enumerate(loader):
            optimizer.zero_grad()
            images = images.to(accelerator.device)
            labels = labels.to(accelerator.device)
            outputs = model(images)
            classes = torch.argmax(outputs, dim=1)
            loss = loss_function(outputs, labels)
            accelerator.backward(loss)
            optimizer.step()
            scheduler.step()
            if (batch%1 == 0) and ('RANK' not in os.environ or os.environ['RANK'] == '0'):
                print('--- Epoch %2i, Batch %3i: Loss = %0.2f ---' % (epoch, batch, loss,))

        if 'RANK' not in os.environ or os.environ['RANK'] == '0' :
            end = time.time()
            imgsec = image_number/(end-start)
            print('--- Epoch %2i, Finished: %0.2f img/sec ---' % (epoch, imgsec))

if __name__ == '__main__':
    main()
```

Aplikacija na više čvorova

accelerate-multinode-run.sh

```
    #!/bin/bash

#PBS -q gpu
#PBS -l select=2:ngpus=2:ncpus=8

# env
module load scientific/pytorch/2.0.0

# cd
cd ${PBS_O_WORKDIR:-""}

# run
accelerate-multinode.sh accelerate-multinode.py
```

accelerate-multinode.py

```
    # source
# - https://github.com/horovod/horovod/blob/master/examples/pytorch/pytorch_synthetic_benchmark.py

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from accelerate import Accelerator

from torchvision import models
from torch.utils.data import DataLoader
from torchvision.datasets import FakeData
from torchvision.transforms import ToTensor

import os
import sys
import time
import pprint
import socket
import numpy as np

def main():

    # settings
    epochs = 10
    batch_size = 256
    image_number = 256*30
    model = 'resnet50'

    # accelerator
    accelerator = Accelerator()

    # model
    model = getattr(models, model)()
    model.to(accelerator.device)

    # optimizer
    optimizer = optim.SGD(model.parameters(), lr=0.01)
    loss_function = nn.CrossEntropyLoss()

    # loader
    data = FakeData(image_number,
                    num_classes=1000,
                    transform=ToTensor())

    loader = DataLoader(data,
                        batch_size=batch_size)
```

```
    # scheduler
    scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9)

    # prepare
    model, optimizer, loader, scheduler = accelerator.prepare(model,
                                                              optimizer,
                                                              loader,
                                                              scheduler)
    # fit
    for epoch in range(epochs):
        start = time.time()
        for batch, (images, labels) in enumerate(loader):
            optimizer.zero_grad()
            images = images.to(accelerator.device)
            labels = labels.to(accelerator.device)
            outputs = model(images)
            classes = torch.argmax(outputs, dim=1)
            loss = loss_function(outputs, labels)
            accelerator.backward(loss)
            optimizer.step()
            scheduler.step()
            if (batch%1 == 0) and ('RANK' not in os.environ or os.environ['RANK'] == '0'):
                print('--- Epoch %2i, Batch %3i: Loss = %0.2f ---' % (epoch, batch, loss,))

        if 'RANK' not in os.environ or os.environ['RANK'] == '0' :
            end = time.time()
            imgsec = image_number/(end-start)
            print('--- Epoch %2i, Finished: %0.2f img/sec ---' % (epoch, imgsec))

if __name__ == '__main__':
    main()
```

Vrančić

Ispod se nalazi primjer aplikacije umjetnog benchmarka koji testira performanse na modelu Resnet50.

Aplikacija na jednom čvoru

singlenode.sh

```
    #PBS -q cpu
#PBS -l ncpus=32
#PBS -l mem=50GB

# environment
module load scientific/pytorch/2.0.0

# set thread number to the cpu one
export OMP_NUM_THREADS=${NCPUS}

# run
cd ${PBS_O_WORKDIR:-""}
python singlenode.py
```

singlenode.py

```
    import os
import time

import torch
import torch.nn as nn
import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision.models import resnet50
from torchvision.datasets import FakeData
from torchvision.transforms import ToTensor

def main():

    # vars
    batch = 16
    samples = 16*30
    epochs = 3

    # model
    model = resnet50(weights=None)
    optimizer = optim.SGD(model.parameters(), lr=0.001)
    loss_fn = nn.CrossEntropyLoss()

    # data
    dataset = FakeData(samples,
                       num_classes=1000,
                       transform=ToTensor())
    loader = DataLoader(dataset,
                        batch_size=batch,
                        shuffle=False,
                        num_workers=1,
                        pin_memory=True)

    # train
    for epoch in range(epochs):
        start = time.time()
        for batch, (images, labels) in enumerate(loader):
            outputs = model(images)
            classes = torch.argmax(outputs, dim=1)
            loss = loss_fn(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            if (batch%10 == 0):
                print('--- Epoch %i, Batch %3i / %3i, Loss = %0.2f ---' % (epoch,
                                                                            batch,
                                                                            len(loader),
                                                                            loss.item()))
        elapsed = time.time()-start
        imgsec = samples/elapsed
        print('--- Epoch %i finished: %0.2f img/sec ---' % (epoch,
                                                            imgsec))

if __name__ == "__main__":
    main()
```

Napomene

Apptainer i run-singlenode.sh

Ova knjižnica je dostavljena u obliku kontejnera, zbog opterećenja koje pip/conda virtualna okruženja stvaraju na Lustre dijeljenim datotečnim sustavima.

Za ispravno izvršavanje python aplikacija ili naredbi koje se u njemu nalaze, potrebno je koristiti wrappere u skriptama sustava PBS:

- Za izvršavanje naredbi u kontejneru na samo jednom čvoru:
  - **run-command.sh**
- Za izvršavanje skripti python na jednom grafičkom procesoru
  - **run-singlegpu.sh**
- Za izvršavanje skripti python na više grafičkih procesora (dostupno za **PyTorch v1.10+**)
  - torchrun/distributed
    - **torchrun-singlenode.sh, distributed-singlenode.sh** - jedan čvor
    - **torchrun-multinode.sh, distributed-multinode.sh** - više čvorova
  - accelerate
    - **accelerate-singlenode.sh** - jedan čvor
    - **accelerate-multinode.sh** - više čvorova

Načini pozivanja wrappera opisani su u primjerima iznad.

Korištenje više grafičkih procesora

PyTorch **ne osigurava** automatsko raspodjeljivanje računa na više grafičkih procesora.

Pri korištenju više procesora, potrebno je koristiti PyTorch sučelje distributed kako je navedeno u primjerima iznad.

U slučaju da vam je ova funkcionalnost potrebna, kontaktirajte nas na computing@srce.hr