Emulate a multi-node setup using just a single node

The goal is to emulate a 2-node environment using a single node with 2 GPUs (for testing purposes). This, of course, can be further expanded to larger set ups.

We use the `deepspeed` launcher here. There is no need to actually use any of the deepspeed code, it's just easier to use its more advanced capabilities. You will just need to install `pip install deepspeed`.

The full setup instructions follow:

1. Create a `hostfile`:

```
$ cat hostfile
worker-0 slots=1
worker-1 slots=1
```

2. Add a matching config to your ssh client

```
$ cat ~/.ssh/config
[...]

Host worker-0
    HostName localhost
    Port 22
Host worker-1
    HostName localhost
    Port 22
```

Adapt the port if it's not 22 and the hostname if `localhost` isn't it.

3. As your local setup is probably password protected ensure to add your public key to `~/.ssh/authorized_keys`

The `deepspeed` launcher explicitly uses no-password connection, e.g. on worker0 it'd run: `ssh -o PasswordAuthentication=no worker-0 hostname`, so you

can always debug ssh setup using:

```
$ ssh -vvv -o PasswordAuthentication=no worker-0 hostname
```

4. Create a test script to check both GPUs are used.

```
$ cat test1.py
import os
import time
import torch
import deepspeed
import torch.distributed as dist

# critical hack to use the 2nd gpu (otherwise both processes will use gpu0)
if os.environ["RANK"] == "1":
    os.environ["CUDA_VISIBLE_DEVICES"] = "1"

dist.init_process_group("nccl")
local_rank = int(os.environ.get("LOCAL_RANK"))
print(f'{dist.get_rank()=}, {local_rank=}')

x = torch.ones(2**30, device=f"cuda:{local_rank}")
time.sleep(100)
```

Run:

```
$ deepspeed -H hostfile test1.py
[2022-09-08 12:02:15,192] [INFO] [runner.py:415:main] Using IP address of 192.168.0.17 for r
[2022-09-08 12:02:15,192] [INFO] [multinode_runner.py:65:get_cmd] Running on the following w
[2022-09-08 12:02:15,192] [INFO] [runner.py:504:main] cmd = pdsh -S -f 1024 -w worker-0,work
worker-0: [2022-09-08 12:02:16,517] [INFO] [launch.py:136:main] WORLD INFO DICT: {'worker-0'
worker-0: [2022-09-08 12:02:16,517] [INFO] [launch.py:142:main] nnodes=2, num_local_procs=1,
worker-0: [2022-09-08 12:02:16,517] [INFO] [launch.py:155:main] global_rank_mapping=defaultd
worker-0: [2022-09-08 12:02:16,517] [INFO] [launch.py:156:main] dist_world_size=2
worker-0: [2022-09-08 12:02:16,517] [INFO] [launch.py:158:main] Setting CUDA_VISIBLE_DEVICES
worker-1: [2022-09-08 12:02:16,518] [INFO] [launch.py:136:main] WORLD INFO DICT: {'worker-0'
worker-1: [2022-09-08 12:02:16,518] [INFO] [launch.py:142:main] nnodes=2, num_local_procs=1,
worker-1: [2022-09-08 12:02:16,518] [INFO] [launch.py:155:main] global_rank_mapping=defaultd
worker-1: [2022-09-08 12:02:16,518] [INFO] [launch.py:156:main] dist_world_size=2
worker-1: [2022-09-08 12:02:16,518] [INFO] [launch.py:158:main] Setting CUDA_VISIBLE_DEVICES
worker-1: torch.distributed.get_rank()=1, local_rank=0
worker-0: torch.distributed.get_rank()=0, local_rank=0
worker-1: tensor([1., 1., 1.,  ..., 1., 1., 1.], device='cuda:0')
worker-0: tensor([1., 1., 1.,  ..., 1., 1., 1.], device='cuda:0')
```

If the ssh set up works you can run `nvidia-smi` in parallel and observe that both GPUs allocated ~4GB of memory from `torch.ones` call.

Note that the script hacks in `CUDA_VISIBLE_DEVICES` to tell the 2nd process to use gpu1, but it'll be seen as `local_rank==0` in both cases.

5. Finally, let's test that NCCL collectives work as well

Script adapted from torch-distributed-gpu-test.py to just tweak `os.environ["CUDA_VISIBLE_DEVICES"]`

```
$ cat test2.py
import deepspeed
import fcntl
import os
import socket
import time
import torch
import torch.distributed as dist

# a critical hack to use the 2nd GPU by the 2nd process (otherwise both processes will use g
if os.environ["RANK"] == "1":
    os.environ["CUDA_VISIBLE_DEVICES"] = "1"

def printflock(*msgs):
    """ solves multi-process interleaved print problem """
    with open(__file__, "r") as fh:
        fcntl.flock(fh, fcntl.LOCK_EX)
        try:
            print(*msgs)
        finally:
            fcntl.flock(fh, fcntl.LOCK_UN)

local_rank = int(os.environ["LOCAL_RANK"])
torch.cuda.set_device(local_rank)
device = torch.device("cuda", local_rank)
hostname = socket.gethostname()

gpu = f"[{hostname}-{local_rank}]"

try:
    # test distributed
    dist.init_process_group("nccl")
    dist.all_reduce(torch.ones(1).to(device), op=dist.ReduceOp.SUM)
    dist.barrier()
    print(f'{dist.get_rank()=}, {local_rank=}')

    # test cuda is available and can allocate memory
    torch.cuda.is_available()
    torch.ones(1).cuda(local_rank)
```

```
    # global rank
    rank = dist.get_rank()
    world_size = dist.get_world_size()

    printflock(f"{gpu} is OK (global rank: {rank}/{world_size})")

    dist.barrier()
    if rank == 0:
        printflock(f"pt={torch.__version__}, cuda={torch.version.cuda}, nccl={torch.cuda.nc
        printflock(f"device compute capabilities={torch.cuda.get_device_capability()}")
        printflock(f"pytorch compute capabilities={torch.cuda.get_arch_list()}")

except Exception:
    printflock(f"{gpu} is broken")
    raise
```

Run:

```
$ deepspeed -H hostfile test2.py
[2022-09-08 12:07:09,336] [INFO] [runner.py:415:main] Using IP address of 192.168.0.17 for r
[2022-09-08 12:07:09,337] [INFO] [multinode_runner.py:65:get_cmd] Running on the following w
[2022-09-08 12:07:09,337] [INFO] [runner.py:504:main] cmd = pdsh -S -f 1024 -w worker-0,work
worker-0: [2022-09-08 12:07:10,635] [INFO] [launch.py:136:main] WORLD INFO DICT: {'worker-0'
worker-0: [2022-09-08 12:07:10,635] [INFO] [launch.py:142:main] nnodes=2, num_local_procs=1,
worker-0: [2022-09-08 12:07:10,635] [INFO] [launch.py:155:main] global_rank_mapping=defaultd
worker-0: [2022-09-08 12:07:10,635] [INFO] [launch.py:156:main] dist_world_size=2
worker-0: [2022-09-08 12:07:10,635] [INFO] [launch.py:158:main] Setting CUDA_VISIBLE_DEVICES
worker-1: [2022-09-08 12:07:10,635] [INFO] [launch.py:136:main] WORLD INFO DICT: {'worker-0'
worker-1: [2022-09-08 12:07:10,635] [INFO] [launch.py:142:main] nnodes=2, num_local_procs=1,
worker-1: [2022-09-08 12:07:10,635] [INFO] [launch.py:155:main] global_rank_mapping=defaultd
worker-1: [2022-09-08 12:07:10,635] [INFO] [launch.py:156:main] dist_world_size=2
worker-1: [2022-09-08 12:07:10,635] [INFO] [launch.py:158:main] Setting CUDA_VISIBLE_DEVICES
worker-0: dist.get_rank()=0, local_rank=0
worker-1: dist.get_rank()=1, local_rank=0
worker-0: [hope-0] is OK (global rank: 0/2)
worker-1: [hope-0] is OK (global rank: 1/2)
worker-0: pt=1.12.1+cu116, cuda=11.6, nccl=(2, 10, 3)
worker-0: device compute capabilities=(8, 0)
worker-0: pytorch compute capabilities=['sm_37', 'sm_50', 'sm_60', 'sm_70', 'sm_75', 'sm_80'
worker-1: [2022-09-08 12:07:13,642] [INFO] [launch.py:318:main] Process 576485 exits success
worker-0: [2022-09-08 12:07:13,642] [INFO] [launch.py:318:main] Process 576484 exits success
```

Voila, missing accomplished.

We tested that the NCCL collectives work, but they use local NVLink/PCIe and not the IB/ETH connections like in real multi-node, so it may or may not be good enough for testing depending on what needs to be tested.

Larger set ups

Now, let's say you have 4 GPUs and you want to emulate 2x2 nodes. Then simply change the `hostfile` to be:

```
$ cat hostfile
worker-0 slots=2
worker-1 slots=2
```

and the `CUDA_VISIBLE_DEVICES` hack to:

```
if os.environ["RANK"] in ["2", "3"]:
    os.environ["CUDA_VISIBLE_DEVICES"] = "2,3"
```

Everything else should be the same.

Automating the process

If you want an automatic approach to handle any shape of topology, you could use something like this:

```
def set_cuda_visible_devices():
    """
    automatically assign the correct groups of gpus for each emulated node by tweaking the
    CUDA_VISIBLE_DEVICES env var
    """

    global_rank = int(os.environ["RANK"])
    world_size = int(os.environ["WORLD_SIZE"])
    emulated_node_size = int(os.environ["LOCAL_SIZE"])
    emulated_node_rank = int(global_rank // emulated_node_size)
    gpus = list(map(str, range(world_size)))
    emulated_node_gpus = ",".join(gpus[emulated_node_rank*emulated_node_size:(emulated_node_
    print(f"Setting CUDA_VISIBLE_DEVICES={emulated_node_gpus}")
    os.environ["CUDA_VISIBLE_DEVICES"] = emulated_node_gpus

set_cuda_visible_devices()
```

Emulating multiple GPUs with a single GPU

The following is an orthogonal need to the one discussed in this document, but it's related so I thought it'd be useful to share some insights here:

With NVIDIA A100 you can use MIG to emulate up to 7 instances of GPUs on just one real GPU, but alas you can't use those instances for anything but standalone use - e.g. you can't do DDP or any NCCL comms over those GPUs. I hoped I could use my A100 to emulate 7 instances and add one more real

GPU and to have 8x GPUs to do development with - but nope it doesn't work. Asking NVIDIA engineers about it, there are no plans to have this use-case supported.

Acknowledgements

Many thanks to Jeff Rasley for helping me to set this up.