BMTrain

BMTrain Team

Aug 17, 2023

GETTING STARTED

1 Installation **3**
- 1.1 Install BMTrain . . . 3
- 1.2 Compilation Options . . . 3
- 1.3 Recommended Configuration . . . 4
- 1.4 FAQ . . . 4

2 Quick Start **5**
- 2.1 Step 1: Initialize BMTrain . . . 5
- 2.2 Step 2: Enable ZeRO-3 Optimization . . . 5
- 2.3 Step 3: Enable Communication Optimization . . . 6
- 2.4 Step 4: Launch Distributed Training . . . 7
- 2.5 Other Notes . . . 8

3 Introduction to Core Technology **9**
- 3.1 ZeRO-3 Optimization . . . 9
- 3.2 Overlap Communication and Computation . . . 9
- 3.3 CPU Offload . . . 10

4 bmtrain **11**
- 4.1 Initialization . . . 11
- 4.2 Distributed Parameters and Modules . . . 11
- 4.3 Methods for Parameters . . . 11
- 4.4 Utilities . . . 11

5 bmtrain.nccl **13**

6 bmtrain.inspect **15**

7 bmtrain.lr_scheduler **17**
- 7.1 LR Schedulers . . . 17

8 API **19**

BMTrain is an efficient large model training toolkit that can be used to train large models with tens of billions of parameters. It can train models in a distributed manner while keeping the code as simple as stand-alone training.

CHAPTER

ONE

INSTALLATION

1.1 Install BMTrain

1.1.1 1. From PyPI (Recommend)

```
$ pip install bmtrain
```

1.1.2 2. From Source

```
$ git clone https://github.com/OpenBMB/BMTrain.git
$ cd BMTrain
$ python3 setup.py install
```

1.2 Compilation Options

By setting environment variables, you can configure the compilation options of BMTrain (by default, the compilation environment will be automatically adapted).

1.2.1 AVX Instructions

- Force the use of AVX instructions: `BMT_AVX256=ON`
- Force the use of AVX512 instructions: `BMT_AVX512=ON`

1.2.2 CUDA Compute Capability

`TORCH_CUDA_ARCH_LIST=6.0 6.1 7.0 7.5 8.0+PTX`

1.3 Recommended Configuration

- NetworkInfiniband 100Gbps / RoCE 100Gbps
- GPUNVIDIA Tesla V100 / NVIDIA Tesla A100 / RTX 3090
- CPUCPU that supports AVX512 instructions, 32 cores or above
- RAM256GB or above

1.4 FAQ

If the following error message is reported during compilation, try using a newer version of the gcc compiler.

```
error: invalid static_cast from type `const torch::OrderdDict<...>`
```

CHAPTER

TWO

QUICK START

2.1 Step 1: Initialize BMTrain

Before you can use BMTrain, you need to initialize it at the beginning of your code. Just like using the distributed module of PyTorch requires the use of **init_process_group** at the beginning of the code, using BMTrain requires the use of **init_distributed** at the beginning of the code.

```
import bmtrain as bmt
bmt.init_distributed(
    seed=0,
    # ...
)
```

NOTE: Do not use PyTorch's distributed module and its associated communication functions when using BMTrain.

2.2 Step 2: Enable ZeRO-3 Optimization

To enable ZeRO-3 optimization, you need to make some simple replacements to the original model's code.

- `torch.nn.Module` -> `bmtrain.DistributedModule`
- `torch.nn.Parameter` -> `bmtrain.DistributedParameter`

And wrap the transformer blocks with `bmtrain.CheckpointBlock`.

Here is an example.

Original

```
import torch
class MyModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.param = torch.nn.Parameter(torch.empty(1024))
        self.module_list = torch.nn.ModuleList([
            SomeTransformerBlock(),
            SomeTransformerBlock(),
            SomeTransformerBlock()
        ])

    def forward(self):
```

(continues on next page)

(continued from previous page)

```
        x = self.param
        for module in self.module_list:
            x = module(x, 1, 2, 3)
        return x
```

Replaced

```
import torch
import bmtrain as bmt
class MyModule(bmt.DistributedModule): # changed here
    def __init__(self):
        super().__init__()
        self.param = bmt.DistributedParameter(torch.empty(1024)) # changed here
        self.module_list = torch.nn.ModuleList([
            bmt.CheckpointBlock(SomeTransformerBlock()), # changed here
            bmt.CheckpointBlock(SomeTransformerBlock()), # changed here
            bmt.CheckpointBlock(SomeTransformerBlock())  # changed here
        ])

    def forward(self):
        x = self.param
        for module in self.module_list:
            x = module(x, 1, 2, 3)
        return x
```

2.3 Step 3: Enable Communication Optimization

To further reduce the extra overhead of communication and overlap communication with computing time, `TransformerBlockList` can be used for optimization.

You can enable them by making the following substitutions to the code:

- `torch.nn.ModuleList` -> `bmtrain.TransformerBlockList`
- `for module in self.module_list: x = module(x, ...)` -> `x = self.module_list(x, ...)`

Original

```
import torch
import bmtrain as bmt
class MyModule(bmt.DistributedModule):
    def __init__(self):
        super().__init__()
        self.param = bmt.DistributedParameter(torch.empty(1024))
        self.module_list = torch.nn.ModuleList([
            bmt.CheckpointBlock(SomeTransformerBlock()),
            bmt.CheckpointBlock(SomeTransformerBlock()),
            bmt.CheckpointBlock(SomeTransformerBlock())
        ])

    def forward(self):
```

(continues on next page)

(continued from previous page)

```
        x = self.param
        for module in self.module_list:
            x = module(x, 1, 2, 3)
        return x
```

Replaced

```
import torch
import bmtrain as bmt
class MyModule(bmt.DistributedModule):
    def __init__(self):
        super().__init__()
        self.param = bmt.DistributedParameter(torch.empty(1024))
        self.module_list = bmt.TransformerBlockList([ # changed here
            bmt.CheckpointBlock(SomeTransformerBlock()),
            bmt.CheckpointBlock(SomeTransformerBlock()),
            bmt.CheckpointBlock(SomeTransformerBlock())
        ])

    def forward(self):
        x = self.param
        x = self.module_list(x, 1, 2, 3) # changed here
        return x
```

2.4 Step 4: Launch Distributed Training

BMTrain uses the same launch command as the distributed module of PyTorch.

You can choose one of them depending on your version of PyTorch.

- `${MASTER_ADDR}` means the IP address of the master node.
- `${MASTER_PORT}` means the port of the master node.
- `${NNODES}` means the total number of nodes.
- `${GPU_PER_NODE}` means the number of GPUs per node.
- `${NODE_RANK}` means the rank of this node.

2.4.1 torch.distributed.launch

```
$ python3 -m torch.distributed.launch --master_addr ${MASTER_ADDR} --master_port $
↪{MASTER_PORT} --nproc_per_node ${GPU_PER_NODE} --nnodes ${NNODES} --node_rank ${NODE_
↪RANK} train.py
```

2.4.2 torchrun

```
$ torchrun --nnodes=${NNODES} --nproc_per_node=${GPU_PER_NODE} --rdzv_id=1 --rdzv_
↪backend=c10d --rdzv_endpoint=${MASTER_ADDR}:${MASTER_PORT} train.py
```

For more information, please refer to the documentation.

2.5 Other Notes

`BMTrain` makes underlying changes to PyTorch, so if your program outputs unexpected results, you can submit information about it in an issue.

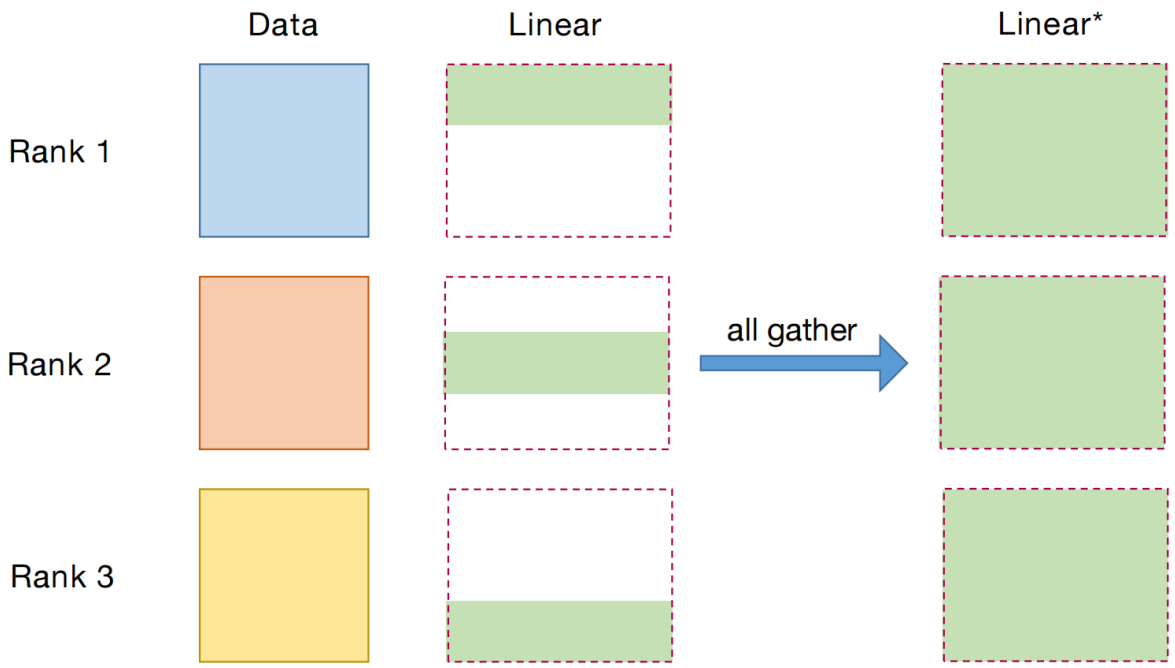
For more examples, please refer to the *examples* folder.

CHAPTER

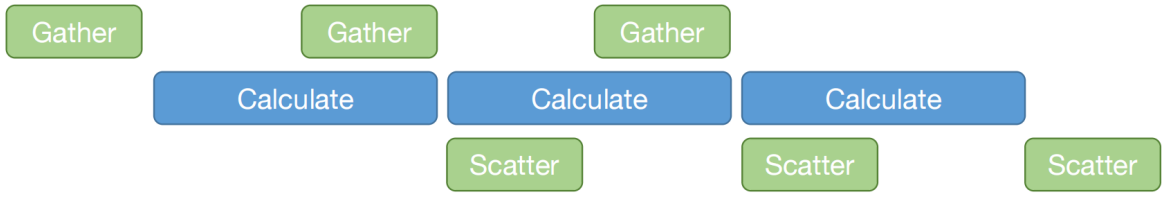
THREE

INTRODUCTION TO CORE TECHNOLOGY

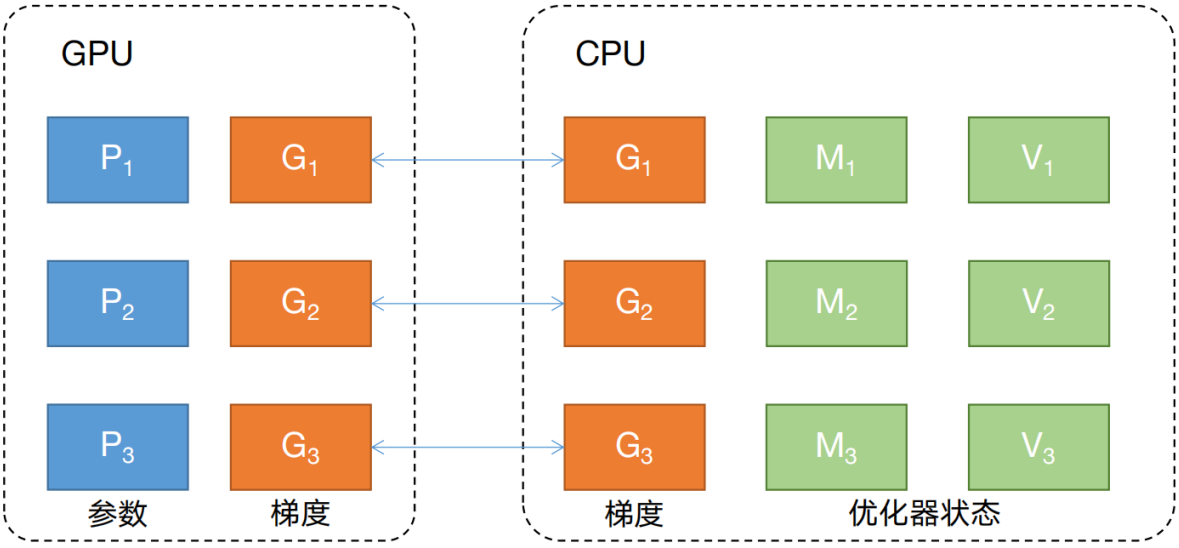
3.1 ZeRO-3 Optimization



3.2 Overlap Communication and Computation



3.3 CPU Offload



CHAPTER

FOUR

BMTRAIN

4.1 Initialization

4.2 Distributed Parameters and Modules

4.3 Methods for Parameters

4.4 Utilities

CHAPTER

FIVE

BMTRAIN.NCCL

CHAPTER

SIX

BMTRAIN.INSPECT

The *bmtrain.inspect* module is a module for debugging and analysis of distributed code.

We recommend that you use the tools in this module to obtain the parameters and computing results in distributed models.

CHAPTER

SEVEN

BMTRAIN.LR_SCHEDULER

The *bmtrain.lr_scheduler* module provides the common learning rate schedulers for big model training.

7.1 LR Schedulers

CHAPTER

EIGHT

API

- genindex
- modindex
- search