热切换接口设计

1、新增接口

```
enum class RunLevel : int8_t {
  UPDATE = 0,
  GRAD,
  ALLOC,
  TOPO
};
```

- `UPDATE` 表示正常的完整运行一次并更新。
- `GRAD` 表示只算出梯度但并不更新。
- `ALLOC` 表示只分配出parameter（如果开了AMP还会分配出data transfer后的parameter）而不进行后续运算。
- `TOPO` 表示只算出exec graph可以cache下来的拓扑但并不进行显存分配与后续运算。

`run()` 函数需要传入 `strategy_id` 和 `run_level`，默认情况 `strategy_id` 为0，`run_level` 为 `RunLevel::UPDATE`。

```
std::shared_ptr<ParamBuffer> _origin_param_buffer;
std::shared_ptr<ParamBuffer> _transfer_param_buffer;
std::shared_ptr<ParamBuffer> _current_grad_buffer;
std::shared_ptr<ParamBuffer> _accumulate_grad_param_buffer;
```

同时，给exec graph增设四种buffer，不同的run level以及切换会用到的不同的buffer。

2、示例

2.1、正常的训练

使用原接口即可，其效果等价于

```
# round 1
run(..., strategy_id = 0, run_level = "update")
# round 2
run(..., strategy_id = 0, run_level = "update")
...
```

2.2、热切换训练

```
# round 1
run(..., strategy_id = 0, run_level = "update")
# round 2
run(..., strategy_id = 1, run_level = "update")
...
```

注意，`strategy_id` 不变但 `fetches` 或者 `feed_dict` 的shape发生改变同样会生成新的exec graph，而导致两次run之间发生热切换。

2.3、多次累积梯度后进行一次更新

```
# round 1
run(..., strategy_id = 0, run_level = "grad")
run(..., strategy_id = 0, run_level = "grad")
...
run(..., strategy_id = 0, run_level = "grad")
run(..., strategy_id = 0, run_level = "update")
# round 2
run(..., strategy_id = 0, run_level = "grad")
run(..., strategy_id = 0, run_level = "grad")
...
run(..., strategy_id = 0, run_level = "grad")
run(..., strategy_id = 0, run_level = "update")
```

2.4、多次累积梯度加热切换后进行一次更新

```
# round 1
run(..., strategy_id = 0, run_level = "alloc")
run(..., strategy_id = 1, run_level = "grad")
run(..., strategy_id = 2, run_level = "grad")
...
run(..., strategy_id = 3, run_level = "grad")
run(..., strategy_id = 0, run_level = "update")
# round 2
run(..., strategy_id = 0, run_level = "alloc")
run(..., strategy_id = 1, run_level = "grad")
run(..., strategy_id = 2, run_level = "grad")
...
run(..., strategy_id = 3, run_level = "grad")
run(..., strategy_id = 0, run_level = "update")
```

其中，第一次`alloc`会让`fp32`的参数保留在该次的exec graph中，在开启AMP的情形下，第二次的`grad`的exec graph只会切换`bf16`的参数，之后连续的几次切换都是如此，最后切回到第一次的exec graph上进行更新即可。之后进入下一轮，仍然需要调用一次`alloc`，因为需要将更新后的`fp32`的参数再次转化成`bf16`的参数，之后操作同理。

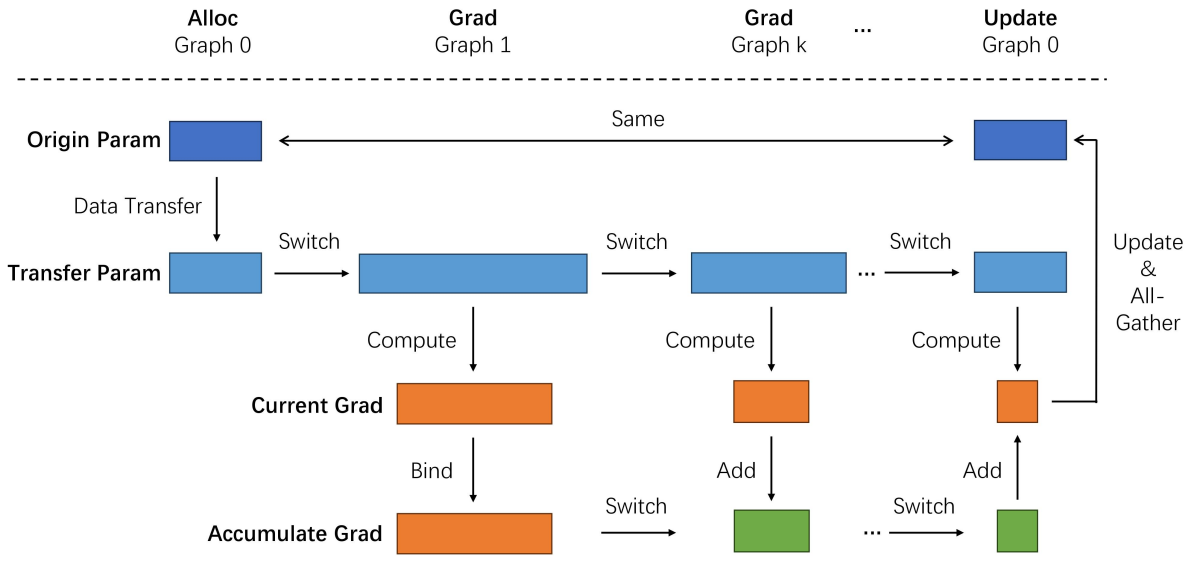
2.5、提前计算拓扑以及热切换的拓扑

```
# topo
run(..., strategy_id = 0, run_level = "topo")
run(..., strategy_id = 1, run_level = "topo")
run(..., strategy_id = 2, run_level = "topo")
...
run(..., strategy_id = 3, run_level = "topo")
run(..., strategy_id = 0, run_level = "topo")
# round 1
run(..., strategy_id = 0, run_level = "alloc")
run(..., strategy_id = 1, run_level = "grad")
run(..., strategy_id = 2, run_level = "grad")
...
run(..., strategy_id = 3, run_level = "grad")
run(..., strategy_id = 0, run_level = "update")
```

```
# round 2
run(..., strategy_id = 0, run_level = "alloc")
run(..., strategy_id = 1, run_level = "grad")
run(..., strategy_id = 2, run_level = "grad")
...
run(..., strategy_id = 3, run_level = "grad")
run(..., strategy_id = 0, run_level = "update")
```

3、buffer操作细节

我么以**多次累积梯度加热切换后进行一次更新**为例，该情形囊括的buffer操作是最为全面的：



- `Switch`表示调用热切换接口。
- `Bind`表示直接共享内存。
- `Compute`表示按照拓扑正常计算，但是跳过Update和Group算子。

值得一提的是，最后的Update阶段实际上就是正常执行完整的拓扑，只不过中间多插了一个`NDArray::add()`。

4、代码

目前运行该部分的脚本在`/home/gehao/lhy/Hetu-dev/examples/nlp/gpt/scripts/lhy_multi_switch.sh`。

切换param和grad的位置在define graph的run中的末尾部分：

```
_param_switcher_pool[key]->SwitchParams(param_switch_mode, param_switch_level);
_grad_switcher_pool[key]->SwitchParams(grad_switch_mode, grad_switch_level);
```

切换部分的主要代码都在`hetu/graph/switch_exec_graph.cc`和`.h`中。

运行`bash scripts/scripts/lhy_multi_switch.sh`，如果将log level设置成info，可以看到不同阶段粗粒度的memory占用。目前只能支持**1-2B**左右模型的多次热切换训练，主要原因是每个strategy下的activation占用的memory都无法得到复用，memory占用会越来越多，下图是一个示例：

```
# config如下
NUM_LAYERS=${1:-32}
HIDDEN_SIZE=${2:-2560}
NUM_HEADS=${3:-32}

# 运行代码如下（只运行一趟）
run_plan(global_batch_size = 2, seq_len = 32, strategy_id = 4, run_level =
ht.run_level("alloc"))
run_plan(global_batch_size = 16, seq_len = 8, strategy_id = 1, run_level =
ht.run_level("grad"))
run_plan(global_batch_size = 8, seq_len = 64, strategy_id = 2, run_level =
ht.run_level("grad"))
run_plan(global_batch_size = 4, seq_len = 16, strategy_id = 3, run_level =
ht.run_level("grad"))
run_plan(global_batch_size = 2, seq_len = 32, strategy_id = 4, run_level =
ht.run_level("update"))
```

```
local_device: device(type=cuda, index=0), all_devices: DeviceGroup([device(type=cuda, index=0),device
use precision ht.bfloat16
load ds_parallel_config from: ds_parallel_config/dp2_tp2_pp2.json,ds_parallel_config/dp8.json,ds_para
optimizer minimize begin...
optimizer minimize end...
strategy_id = 4, dp_size = 2, seq_len = 32 run begin
[default_define_and_run_executable_0 run begin] Device 0:
Used Memory: 3 MiB
[default_define_and_run_executable_0 run ALLOC end] Device 0:
Used Memory: 5792 MiB
strategy_id = 4, dp_size = 2, seq_len = 32 run end
strategy_id = 1, dp_size = 2, seq_len = 8 run begin
param switch from default_define_and_run_executable_0 to default_define_and_run_executable_1 is done
[default_define_and_run_executable_1 run begin] Device 0:
Used Memory: 17339 MiB
[default_define_and_run_executable_1 run GRAD end] Device 0:
Used Memory: 27191 MiB
strategy_id = 1, dp_size = 2, seq_len = 8 run end
strategy_id = 2, dp_size = 2, seq_len = 64 run begin
param switch from default_define_and_run_executable_1 to default_define_and_run_executable_2 is done
grad switch from default_define_and_run_executable_1 to default_define_and_run_executable_2 is done
[default_define_and_run_executable_2 run begin] Device 0:
Used Memory: 24717 MiB
[default_define_and_run_executable_2 run GRAD end] Device 0:
Used Memory: 32111 MiB
strategy_id = 2, dp_size = 2, seq_len = 64 run end
strategy_id = 3, dp_size = 2, seq_len = 16 run begin
param switch from default_define_and_run_executable_2 to default_define_and_run_executable_3 is done
grad switch from default_define_and_run_executable_2 to default_define_and_run_executable_3 is done
[default_define_and_run_executable_3 run begin] Device 0:
Used Memory: 30255 MiB
[default_define_and_run_executable_3 run GRAD end] Device 0:
Used Memory: 36425 MiB
strategy_id = 3, dp_size = 2, seq_len = 16 run end
strategy_id = 4, dp_size = 2, seq_len = 32 run begin
param switch from default_define_and_run_executable_3 to default_define_and_run_executable_0 is done
grad switch from default_define_and_run_executable_3 to default_define_and_run_executable_0 is done
[default_define_and_run_executable_0 run begin] Device 0:
Used Memory: 35191 MiB
[default_define_and_run_executable_0 run UPDATE end] Device 0:
Used Memory: 36618 MiB
strategy_id = 4, dp_size = 2, seq_len = 32 run end
loss = 9.8125 and time = 7.117199420928955
train hetu ds parallel end...
```

我们可以看到很大一部分显存增长来自于每次运行exec graph的内部，运行一趟后，再想从 `tp=8` 重新切换到 `dp=8` 的过程中就会爆显存。

5、TODO

---年前遗留问题---

- **已解决**：目前部分情形下可能出现bug（非热切换情形下多次累积梯度后进行一次更新会有问题）。
- **已解决**：scale到大参数量的情形下，验证各buffer的显存是否有未释放干净的情形。
- **已解决**：支持动态`seq_len`。
- **已解决**：优化`concatenate`算子。
- **已解决**：解决`tp8`的assert报错。

---新增问题---

- **已解决**：`dp=4、tp=2`切换到`dp=2、tp=4`切换到`tp=8`仍然有通信开销，原因是所有参数的ds的order都是dup在前split在后，这会导致相邻近的rank不具有相同的dup，热切换仍有通信，并不是直接切开就可以。
- **已解决**：为进一步加快切换速率，将所有split 1都改成了split 0，这样热切换的concatenate都可以调memory copy。
- **已解决**：较为复杂的grad切换（对dim 0和dim 1都有slice的grad）存在illegal memory access的bug，目前已解决。
- **待解决**：有时python端会出现numpy库的一些segmentation fault之类的报错（不止一种，有随机性），很奇怪，待定位。
- **待解决**：验证去除current grad buffer，代码已写好，未验证（理论上如果用目前的memory pool，去除掉current grad buffer，当前grad的碎片化的内存实际上无法得到free或者重用，会比使用current grad buffer统一存储grad并释放更占内存，因此这部分的代码还没去验证）。
- **待解决**：热切换效率优化。