TestReduce

March 19, 2024

[1]:
```
import torch
torch.cuda.is_available()
!export LC_ALL="en_US.UTF-8"
!export LD_LIBRARY_PATH="/usr/lib64-nvidia"
!export LIBRARY_PATH="/usr/local/cuda/lib64/stubs"
!ldconfig /usr/lib64-nvidia
```

```
/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_5.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbb.so.12 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc_proxy.so.2 is not a symbolic
link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link
```

[ ]:
```
import torch

def fn(x, y):
    return torch.sum(x, 3, keepdim=True) + y
fnc = torch.compile(fn)

bsz = 4
num_head = 32
seq_len = 2048
head_dim = 128
x = torch.randn([bsz, num_head, seq_len, head_dim]).cuda()
y = torch.randn([bsz, num_head, seq_len, head_dim]).cuda()
z = fnc(x, y)
print(z[0,0,0,0])
```

```
tensor(-4.7261, device='cuda:0')
```

[ ]:
```
!cat "/tmp/torchinductor_root/xi/
↪cxioebgg7ylngwvuclwlpnowzdzo2qa7dt2ajswxgbdzwpqrqlo3.py"
```

```
from ctypes import c_void_p, c_long
import torch
import math
import random
import os
import tempfile
from math import inf, nan
from torch._inductor.hooks import run_intermediate_hooks
from torch._inductor.utils import maybe_profile

from torch import empty_strided, device
from torch._inductor.codecache import AsyncCompile
from torch._inductor.select_algorithm import extern_kernels

aten = torch.ops.aten
assert_size_stride = torch._C._dynamo.guards.assert_size_stride
reinterpret_tensor = torch.ops.inductor._reinterpret_tensor
async_compile = AsyncCompile()


# kernel path: /tmp/torchinductor_root/67/c67jzc7zirk7w5p43jvlmyypjcwsz3lobvbuba
qemjb43aeo4j4c.py
# Source Nodes: [add, sum_1], Original ATen: [aten.add, aten.sum]
# add => add
# sum_1 => sum_1
triton_per_fused_add_sum_0 = async_compile.triton('triton_', '''
import triton
import triton.language as tl
from torch._inductor.ir import ReductionHint
from torch._inductor.ir import TileHint
from torch._inductor.triton_heuristics import AutotuneHint, persistent_reduction
from torch._inductor.utils import instance_descriptor
from torch._inductor import triton_helpers

@persistent_reduction(
    size_hints=[262144, 128],
    reduction_hint=ReductionHint.INNER,
    filename=__file__,
    meta={'signature': {0: '*fp32', 1: '*fp32', 2: '*fp32', 3: 'i32', 4: 'i32'},
'device': 0, 'device_type': 'cuda', 'constants': {}, 'mutated_arg_names': [],
'autotune_hints': set(), 'kernel_name': 'triton_per_fused_add_sum_0', 'configs':
[instance_descriptor(divisible_by_16=(0, 1, 2, 3, 4), equal_to_1=(),
ids_of_folded_args=(), divisible_by_8=(3, 4))]}
```

```
)
@triton.jit
def triton_(in_ptr0, in_ptr1, out_ptr1, xnumel, rnumel, XBLOCK : tl.constexpr):
    xnumel = 262144
    rnumel = 128
    RBLOCK: tl.constexpr = 128
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:, None]
    xmask = xindex < xnumel
    rindex = tl.arange(0, RBLOCK)[None, :]
    rmask = rindex < rnumel
    r1 = rindex
    x0 = xindex
    tmp0 = tl.load(in_ptr0 + (r1 + (128*x0)), rmask, other=0)
    tmp5 = tl.load(in_ptr1 + (r1 + (128*x0)), rmask, other=0)
    tmp1 = tl.broadcast_to(tmp0, [XBLOCK, RBLOCK])
    tmp3 = tl.where(rmask, tmp1, 0)
    tmp4 = tl.sum(tmp3, 1)[:, None]
    tmp6 = tmp4 + tmp5
    tl.store(out_ptr1 + (r1 + (128*x0)), tmp6, rmask)
''')

import triton
import triton.language as tl
from torch._inductor.triton_heuristics import grid, start_graph, end_graph
from torch._C import _cuda_getCurrentRawStream as get_cuda_stream


async_compile.wait(globals())
del async_compile

def call(args):
    arg0_1, arg1_1 = args
    args.clear()
    assert_size_stride(arg0_1, (4, 32, 2048, 128), (8388608, 262144, 128, 1))
    assert_size_stride(arg1_1, (4, 32, 2048, 128), (8388608, 262144, 128, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0) # no-op to ensure context
        buf1 = empty_strided((4, 32, 2048, 128), (8388608, 262144, 128, 1),
device='cuda', dtype=torch.float32)
        # Source Nodes: [add, sum_1], Original ATen: [aten.add, aten.sum]
        stream0 = get_cuda_stream(0)
        triton_per_fused_add_sum_0.run(arg0_1, arg1_1, buf1, 262144, 128,
grid=grid(262144), stream=stream0)
        del arg0_1
        del arg1_1
        return (buf1, )
```

```
def benchmark_compiled_module(times=10, repeat=10):
    from torch._dynamo.testing import rand_strided
    from torch._inductor.utils import print_performance
    arg0_1 = rand_strided((4, 32, 2048, 128), (8388608, 262144, 128, 1),
device='cuda:0', dtype=torch.float32)
    arg1_1 = rand_strided((4, 32, 2048, 128), (8388608, 262144, 128, 1),
device='cuda:0', dtype=torch.float32)
    return print_performance(lambda: call([arg0_1, arg1_1]), times=times,
repeat=repeat)


if __name__ == "__main__":
    from torch._inductor.wrapper_benchmark import compiled_module_main
    compiled_module_main('None', benchmark_compiled_module)
```

[2]:
```
import torch

def fn(x, y):
    return torch.sum(x, 2, keepdim=True) + y
fnc = torch.compile(fn)

bsz = 2
num_head = 2
seq_len = 2048
head_dim = 128
x = torch.randn([bsz, num_head, seq_len, head_dim]).cuda()
y = torch.randn([bsz, num_head, seq_len, head_dim]).cuda()
z = fnc(x, y)
print(z[0,0,0,0])
```

```
tensor(6.2767, device='cuda:0')
```

[3]:
```
!cat "/tmp/torchinductor_root/hr/
 ↪chr7jtgh45ys3gwvhxjmh5eb3jfuuovog5m7xghyb2jtjd5kyjio.py"
```

```
from ctypes import c_void_p, c_long
import torch
import math
import random
import os
import tempfile
from math import inf, nan
from torch._inductor.hooks import run_intermediate_hooks
from torch._inductor.utils import maybe_profile
from torch._inductor.codegen.memory_planning import _align as align
```

```
from torch import device, empty, empty_strided
from torch._inductor.codecache import AsyncCompile
from torch._inductor.select_algorithm import extern_kernels

aten = torch.ops.aten
inductor_ops = torch.ops.inductor
assert_size_stride = torch._C._dynamo.guards.assert_size_stride
alloc_from_pool = torch.ops.inductor._alloc_from_pool
reinterpret_tensor = torch.ops.inductor._reinterpret_tensor
async_compile = AsyncCompile()


# kernel path: /tmp/torchinductor_root/ld/cldqk67bhw4d3ylspigfcwpllyqfbnppr3vfkk
5cnltpjgstp7gv.py
# Source Nodes: [sum_1], Original ATen: [aten.sum]
# sum_1 => sum_1
triton_red_fused_sum_0 = async_compile.triton('triton_', '''
import triton
import triton.language as tl
from torch._inductor.ir import ReductionHint
from torch._inductor.ir import TileHint
from torch._inductor.triton_heuristics import AutotuneHint, reduction
from torch._inductor.utils import instance_descriptor
from torch._inductor import triton_helpers

@reduction(
    size_hints=[8192, 128],
    reduction_hint=ReductionHint.OUTER,
    filename=__file__,
    triton_meta={'signature': {0: '*fp32', 1: '*fp32', 2: 'i32', 3: 'i32'},
'device': 0, 'device_type': 'cuda', 'constants': {}, 'configs':
[instance_descriptor(divisible_by_16=(0, 1, 2, 3), equal_to_1=(),
ids_of_folded_args=(), divisible_by_8=(2, 3))]},
    inductor_meta={'autotune_hints': set(), 'kernel_name':
'triton_red_fused_sum_0', 'mutated_arg_names': []}
)
@triton.jit
def triton_(in_ptr0, out_ptr0, xnumel, rnumel, XBLOCK : tl.constexpr, RBLOCK :
tl.constexpr):
    xnumel = 8192
    rnumel = 128
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:, None]
    xmask = xindex < xnumel
    rbase = tl.arange(0, RBLOCK)[None, :]
    x0 = xindex % 128
    x1 = (xindex // 128)
    _tmp2 = tl.full([XBLOCK, RBLOCK], 0, tl.float32)
```

```
    x3 = xindex
    for roffset in range(0, rnumel, RBLOCK):
        rindex = roffset + rbase
        rmask = rindex < rnumel
        r2 = rindex
        tmp0 = tl.load(in_ptr0 + (x0 + (128*r2) + (16384*x1)), rmask,
eviction_policy='evict_first', other=0.0)
        tmp1 = tl.broadcast_to(tmp0, [XBLOCK, RBLOCK])
        tmp3 = _tmp2 + tmp1
        _tmp2 = tl.where(rmask, tmp3, _tmp2)
    tmp2 = tl.sum(_tmp2, 1)[:, None]
    tl.store(out_ptr0 + (x3), tmp2, None)
''')

import triton
import triton.language as tl
from torch._inductor.triton_heuristics import grid, start_graph, end_graph
from torch._C import _cuda_getCurrentRawStream as get_cuda_stream


# kernel path: /tmp/torchinductor_root/yk/cykivfjr5qfiimyxbmh3ll3owyixse33f5zqbu
e46zyytwmxh4ob.py
# Source Nodes: [sum_1], Original ATen: [aten.sum]
# sum_1 => sum_1
triton_per_fused_sum_1 = async_compile.triton('triton_', '''
import triton
import triton.language as tl
from torch._inductor.ir import ReductionHint
from torch._inductor.ir import TileHint
from torch._inductor.triton_heuristics import AutotuneHint, persistent_reduction
from torch._inductor.utils import instance_descriptor
from torch._inductor import triton_helpers

@persistent_reduction(
    size_hints=[512, 16],
    reduction_hint=ReductionHint.OUTER_TINY,
    filename=__file__,
    triton_meta={'signature': {0: '*fp32', 1: '*fp32', 2: 'i32', 3: 'i32'},
'device': 0, 'device_type': 'cuda', 'constants': {}, 'configs':
[instance_descriptor(divisible_by_16=(0, 1, 2, 3), equal_to_1=(),
ids_of_folded_args=(), divisible_by_8=(2, 3))]},
    inductor_meta={'autotune_hints': set(), 'kernel_name':
'triton_per_fused_sum_1', 'mutated_arg_names': []}
)
@triton.jit
def triton_(in_ptr0, out_ptr0, xnumel, rnumel, XBLOCK : tl.constexpr):
    xnumel = 512
    rnumel = 16
```

```
    RBLOCK: tl.constexpr = 16
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:, None]
    xmask = xindex < xnumel
    rindex = tl.arange(0, RBLOCK)[None, :]
    rmask = rindex < rnumel
    r2 = rindex
    x0 = xindex % 128
    x1 = (xindex // 128)
    x3 = xindex
    tmp0 = tl.load(in_ptr0 + (x0 + (128*r2) + (2048*x1)), rmask & xmask,
other=0.0)
    tmp1 = tl.broadcast_to(tmp0, [XBLOCK, RBLOCK])
    tmp3 = tl.where(rmask & xmask, tmp1, 0)
    tmp4 = tl.sum(tmp3, 1)[:, None]
    tl.store(out_ptr0 + (x3), tmp4, xmask)
''')


# kernel path: /tmp/torchinductor_root/s2/cs2tufhqacy2b4nxk2ic46prdelrf3zzqwsciv
ftt4j7tqx3ilsb.py
# Source Nodes: [add], Original ATen: [aten.add]
# add => add
triton_poi_fused_add_2 = async_compile.triton('triton_', '''
import triton
import triton.language as tl
from torch._inductor.ir import ReductionHint
from torch._inductor.ir import TileHint
from torch._inductor.triton_heuristics import AutotuneHint, pointwise
from torch._inductor.utils import instance_descriptor
from torch._inductor import triton_helpers

@pointwise(
    size_hints=[1048576],
    filename=__file__,
    triton_meta={'signature': {0: '*fp32', 1: '*fp32', 2: '*fp32', 3: 'i32'},
'device': 0, 'device_type': 'cuda', 'constants': {}, 'configs':
[instance_descriptor(divisible_by_16=(0, 1, 2, 3), equal_to_1=(),
ids_of_folded_args=(), divisible_by_8=(3,))]},
    inductor_meta={'autotune_hints': set(), 'kernel_name':
'triton_poi_fused_add_2', 'mutated_arg_names': []},
    min_elem_per_thread=0
)
@triton.jit
def triton_(in_ptr0, in_ptr1, out_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 1048576
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
```

```
    xmask = xindex < xnumel
    x0 = xindex % 128
    x2 = (xindex // 262144)
    x3 = xindex
    tmp0 = tl.load(in_ptr0 + (x0 + (128*x2)), None,
eviction_policy='evict_last')
    tmp1 = tl.load(in_ptr1 + (x3), None)
    tmp2 = tmp0 + tmp1
    tl.store(out_ptr0 + (x3), tmp2, None)
''')


async_compile.wait(globals())
del async_compile

def call(args):
    arg0_1, arg1_1 = args
    args.clear()
    assert_size_stride(arg0_1, (2, 2, 2048, 128), (524288, 262144, 128, 1))
    assert_size_stride(arg1_1, (2, 2, 2048, 128), (524288, 262144, 128, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0) # no-op to ensure context
        buf0 = empty_strided((2, 2, 1, 128, 16), (4096, 2048, 8192, 1, 128),
device='cuda', dtype=torch.float32)
        # Source Nodes: [sum_1], Original ATen: [aten.sum]
        stream0 = get_cuda_stream(0)
        triton_red_fused_sum_0.run(arg0_1, buf0, 8192, 128, grid=grid(8192),
stream=stream0)
        del arg0_1
        buf1 = empty_strided((2, 2, 1, 128), (256, 128, 512, 1), device='cuda',
dtype=torch.float32)
        # Source Nodes: [sum_1], Original ATen: [aten.sum]
        triton_per_fused_sum_1.run(buf0, buf1, 512, 16, grid=grid(512),
stream=stream0)
        del buf0
        buf2 = empty((2, 2, 2048, 128), device='cuda', dtype=torch.float32)
        # Source Nodes: [add], Original ATen: [aten.add]
        triton_poi_fused_add_2.run(buf1, arg1_1, buf2, 1048576,
grid=grid(1048576), stream=stream0)
        del arg1_1
        return (buf2, )


def benchmark_compiled_module(times=10, repeat=10):
    from torch._dynamo.testing import rand_strided
    from torch._inductor.utils import print_performance
    arg0_1 = rand_strided((2, 2, 2048, 128), (524288, 262144, 128, 1),
device='cuda:0', dtype=torch.float32)
```

```
    arg1_1 = rand_strided((2, 2, 2048, 128), (524288, 262144, 128, 1),
device='cuda:0', dtype=torch.float32)
    fn = lambda: call([arg0_1, arg1_1])
    return print_performance(fn, times=times, repeat=repeat)


if __name__ == "__main__":
    from torch._inductor.wrapper_benchmark import compiled_module_main
    compiled_module_main('None', benchmark_compiled_module)
```

[2]:
```
import torch

def fn(x, y):
    return torch.sum(x, 0, keepdim=True) + y
fnc = torch.compile(fn)

bsz = 4
num_head = 32
seq_len = 2048
head_dim = 128
x = torch.randn([bsz, num_head, seq_len, head_dim]).cuda()
y = torch.randn([bsz, num_head, seq_len, head_dim]).cuda()
z = fnc(x, y)
print(z[0,0,0,0])
```

```
tensor(-0.4866, device='cuda:0')
```

[3]:
```
!cat "/tmp/torchinductor_root/bq/
 ↪cbqnrx3hiylqaqbfhx5lufog6y6zrn3mwfq4ge73eepu2qbnb4o3.py"
```

```
from ctypes import c_void_p, c_long
import torch
import math
import random
import os
import tempfile
from math import inf, nan
from torch._inductor.hooks import run_intermediate_hooks
from torch._inductor.utils import maybe_profile
from torch._inductor.codegen.memory_planning import _align as align

from torch import device, empty, empty_strided
from torch._inductor.codecache import AsyncCompile
from torch._inductor.select_algorithm import extern_kernels

aten = torch.ops.aten
inductor_ops = torch.ops.inductor
```

```
assert_size_stride = torch._C._dynamo.guards.assert_size_stride
alloc_from_pool = torch.ops.inductor._alloc_from_pool
reinterpret_tensor = torch.ops.inductor._reinterpret_tensor
async_compile = AsyncCompile()


# kernel path: /tmp/torchinductor_root/vv/cvvlr32hd7kdbo7qfsx2kq25uqhx6xdyaxucdq
itbd5is7dxojn3.py
# Source Nodes: [add, sum_1], Original ATen: [aten.add, aten.sum]
# add => add
# sum_1 => sum_1
triton_poi_fused_add_sum_0 = async_compile.triton('triton_', '''
import triton
import triton.language as tl
from torch._inductor.ir import ReductionHint
from torch._inductor.ir import TileHint
from torch._inductor.triton_heuristics import AutotuneHint, pointwise
from torch._inductor.utils import instance_descriptor
from torch._inductor import triton_helpers

@pointwise(
    size_hints=[33554432],
    filename=__file__,
    triton_meta={'signature': {0: '*fp32', 1: '*fp32', 2: '*fp32', 3: 'i32'},
'device': 0, 'device_type': 'cuda', 'constants': {}, 'configs':
[instance_descriptor(divisible_by_16=(0, 1, 2, 3), equal_to_1=(),
ids_of_folded_args=(), divisible_by_8=(3,))]},
    inductor_meta={'autotune_hints': set(), 'kernel_name':
'triton_poi_fused_add_sum_0', 'mutated_arg_names': []},
    min_elem_per_thread=0
)
@triton.jit
def triton_(in_ptr0, in_ptr1, out_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 33554432
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x0 = xindex % 8388608
    x2 = xindex
    tmp0 = tl.load(in_ptr0 + (x0), None, eviction_policy='evict_last')
    tmp1 = tl.load(in_ptr0 + (8388608 + x0), None, eviction_policy='evict_last')
    tmp3 = tl.load(in_ptr0 + (16777216 + x0), None,
eviction_policy='evict_last')
    tmp5 = tl.load(in_ptr0 + (25165824 + x0), None,
eviction_policy='evict_last')
    tmp7 = tl.load(in_ptr1 + (x2), None)
    tmp2 = tmp0 + tmp1
    tmp4 = tmp2 + tmp3
```

```
    tmp6 = tmp4 + tmp5
    tmp8 = tmp6 + tmp7
    tl.store(out_ptr0 + (x2), tmp8, None)
''')

import triton
import triton.language as tl
from torch._inductor.triton_heuristics import grid, start_graph, end_graph
from torch._C import _cuda_getCurrentRawStream as get_cuda_stream


async_compile.wait(globals())
del async_compile

def call(args):
    arg0_1, arg1_1 = args
    args.clear()
    assert_size_stride(arg0_1, (4, 32, 2048, 128), (8388608, 262144, 128, 1))
    assert_size_stride(arg1_1, (4, 32, 2048, 128), (8388608, 262144, 128, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0) # no-op to ensure context
        buf0 = empty((4, 32, 2048, 128), device='cuda', dtype=torch.float32)
        # Source Nodes: [add, sum_1], Original ATen: [aten.add, aten.sum]
        stream0 = get_cuda_stream(0)
        triton_poi_fused_add_sum_0.run(arg0_1, arg1_1, buf0, 33554432,
grid=grid(33554432), stream=stream0)
        del arg0_1
        del arg1_1
        return (buf0, )


def benchmark_compiled_module(times=10, repeat=10):
    from torch._dynamo.testing import rand_strided
    from torch._inductor.utils import print_performance
    arg0_1 = rand_strided((4, 32, 2048, 128), (8388608, 262144, 128, 1),
device='cuda:0', dtype=torch.float32)
    arg1_1 = rand_strided((4, 32, 2048, 128), (8388608, 262144, 128, 1),
device='cuda:0', dtype=torch.float32)
    fn = lambda: call([arg0_1, arg1_1])
    return print_performance(fn, times=times, repeat=repeat)


if __name__ == "__main__":
    from torch._inductor.wrapper_benchmark import compiled_module_main
    compiled_module_main('None', benchmark_compiled_module)
```