TestBroadcast

March 19, 2024

[ ]:
```
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.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/libtbbmalloc.so.2 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/libtbb.so.12 is not a symbolic link
```

[ ]:
```
import torch

def fn(x, y):
    return x + y

fnc = torch.compile(fn)
x = torch.randn([16,32,1]).cuda()
y = torch.randn([16,32,64]).cuda()
z = fnc(x, y)
print(z[0,0,0])
```

```
tensor(2.3866, device='cuda:0')
```

[ ]:
```
!cat "/tmp/torchinductor_root/dk/
 ↪cdk5k4pkaaigmex3d2z377tog33hk2enoa5pajkz2bse4qlzy7ol.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/qj/cqja774gf4im47q4qbhed6h7nsp2uyfe4hiwkt
tkebk2aq2l6dye.py
# Source Nodes: [add], Original ATen: [aten.add]
# add => add
triton_poi_fused_add_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=[32768],
    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_0', 'mutated_arg_names': []},
    min_elem_per_thread=0
)
@triton.jit
```

```
def triton_(in_ptr0, in_ptr1, out_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 32768
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x1 = (xindex // 64)
    x2 = xindex
    tmp0 = tl.load(in_ptr0 + (x1), None, eviction_policy='evict_last')
    tmp1 = tl.load(in_ptr1 + (x2), None)
    tmp2 = tmp0 + tmp1
    tl.store(out_ptr0 + (x2), 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


async_compile.wait(globals())
del async_compile

def call(args):
    arg0_1, arg1_1 = args
    args.clear()
    assert_size_stride(arg0_1, (16, 32, 1), (32, 1, 1))
    assert_size_stride(arg1_1, (16, 32, 64), (2048, 64, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0) # no-op to ensure context
        buf0 = empty((16, 32, 64), device='cuda', dtype=torch.float32)
        # Source Nodes: [add], Original ATen: [aten.add]
        stream0 = get_cuda_stream(0)
        triton_poi_fused_add_0.run(arg0_1, arg1_1, buf0, 32768,
grid=grid(32768), 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((16, 32, 1), (32, 1, 1), device='cuda:0',
dtype=torch.float32)
    arg1_1 = rand_strided((16, 32, 64), (2048, 64, 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)
```

[ ]:
```
import torch

def fn(x, y):
    return x + y

fnc = torch.compile(fn)
x = torch.randn([16,1,64]).cuda()
y = torch.randn([16,32,64]).cuda()
z = fnc(x, y)
print(z[0,0,0])
```

```
tensor(0.1840, device='cuda:0')
```

[ ]:
```
!cat "/tmp/torchinductor_root/k4/
 ↪ck4ew4lwecyh4hkvllbjuewsjnexqnx7be73fbypjjlw4h5akfxh.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/4f/c4fx6cn6fztxho36nzcxhy3gcfxccgluwh5k3z
cduobz46tgugto.py
```

```
# Source Nodes: [add], Original ATen: [aten.add]
# add => add
triton_poi_fused_add_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=[32768],
    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_0', 'mutated_arg_names': []},
    min_elem_per_thread=0
)
@triton.jit
def triton_(in_ptr0, in_ptr1, out_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 32768
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x0 = xindex % 64
    x2 = (xindex // 2048)
    x3 = xindex
    tmp0 = tl.load(in_ptr0 + (x0 + (64*x2)), None, eviction_policy='evict_last')
    tmp1 = tl.load(in_ptr1 + (x3), None)
    tmp2 = tmp0 + tmp1
    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


async_compile.wait(globals())
del async_compile

def call(args):
    arg0_1, arg1_1 = args
```

```
    args.clear()
    assert_size_stride(arg0_1, (16, 1, 64), (64, 64, 1))
    assert_size_stride(arg1_1, (16, 32, 64), (2048, 64, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0) # no-op to ensure context
        buf0 = empty((16, 32, 64), device='cuda', dtype=torch.float32)
        # Source Nodes: [add], Original ATen: [aten.add]
        stream0 = get_cuda_stream(0)
        triton_poi_fused_add_0.run(arg0_1, arg1_1, buf0, 32768,
grid=grid(32768), 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((16, 1, 64), (64, 64, 1), device='cuda:0',
dtype=torch.float32)
    arg1_1 = rand_strided((16, 32, 64), (2048, 64, 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)
```

[ ]:
```
import torch

def fn(x, y):
    return x + y

fnc = torch.compile(fn)
x = torch.randn([1,32,64]).cuda()
y = torch.randn([16,32,64]).cuda()
z = fnc(x, y)
print(z[0,0,0])
```

```
tensor(-0.1861, device='cuda:0')
```

[ ]:
```
!cat "/tmp/torchinductor_root/tl/
 ↪ctlg54y25n346hgkhr6judu4e7d4igki7f64adnipi4yybsdv7rt.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/ly/clyadqqk2cm34cj6wtvktre2fvtmujjsk7u2jc
g7wxhxhm65kfc2.py
# Source Nodes: [add], Original ATen: [aten.add]
# add => add
triton_poi_fused_add_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=[32768],
    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_0', 'mutated_arg_names': []},
    min_elem_per_thread=0
)
@triton.jit
def triton_(in_ptr0, in_ptr1, out_ptr0, xnumel, XBLOCK : tl.constexpr):
```

```
    xnumel = 32768
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x0 = xindex % 2048
    x2 = xindex
    tmp0 = tl.load(in_ptr0 + (x0), None, eviction_policy='evict_last')
    tmp1 = tl.load(in_ptr1 + (x2), None)
    tmp2 = tmp0 + tmp1
    tl.store(out_ptr0 + (x2), 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


async_compile.wait(globals())
del async_compile

def call(args):
    arg0_1, arg1_1 = args
    args.clear()
    assert_size_stride(arg0_1, (1, 32, 64), (2048, 64, 1))
    assert_size_stride(arg1_1, (16, 32, 64), (2048, 64, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0) # no-op to ensure context
        buf0 = empty((16, 32, 64), device='cuda', dtype=torch.float32)
        # Source Nodes: [add], Original ATen: [aten.add]
        stream0 = get_cuda_stream(0)
        triton_poi_fused_add_0.run(arg0_1, arg1_1, buf0, 32768,
grid=grid(32768), 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((1, 32, 64), (2048, 64, 1), device='cuda:0',
dtype=torch.float32)
    arg1_1 = rand_strided((16, 32, 64), (2048, 64, 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)
```

[ ]:
```
import torch

def fn(x, y):
    return x + y

fnc = torch.compile(fn)
x = torch.randn([1024,32,1]).cuda()
y = torch.randn([1,32,128]).cuda()
z = fnc(x, y)
print(z[0,0,0])
```

```
tensor(3.1080, device='cuda:0')
```

[ ]:
```
!cat "/tmp/torchinductor_root/5e/
 ↪c5ekhq632y6sg2arf2fczrp7bweenywsr2x2ihy7rfcx3dusjobg.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/qi/cqiwmqtstrkuky6xlcsj2xrk5cihj6td5g2yzz
kv3haysjwjtmr3.py
# Source Nodes: [add], Original ATen: [aten.add]
# add => add
triton_poi_fused_add_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=[4194304], filename=__file__, meta={'signature': {0:
'*fp32', 1: '*fp32', 2: '*fp32', 3: 'i32'}, 'device': 0, 'device_type': 'cuda',
'constants': {}, 'mutated_arg_names': [], 'autotune_hints': set(),
'kernel_name': 'triton_poi_fused_add_0', 'configs':
[instance_descriptor(divisible_by_16=(0, 1, 2, 3), equal_to_1=(),
ids_of_folded_args=(), divisible_by_8=(3,))]})
@triton.jit
def triton_(in_ptr0, in_ptr1, out_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 4194304
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x3 = (xindex // 128)
    x4 = xindex % 4096
    x5 = xindex
    tmp0 = tl.load(in_ptr0 + (x3), None, eviction_policy='evict_last')
    tmp1 = tl.load(in_ptr1 + (x4), None, eviction_policy='evict_last')
    tmp2 = tmp0 + tmp1
    tl.store(out_ptr0 + (x5), 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


async_compile.wait(globals())
del async_compile

def call(args):
    arg0_1, arg1_1 = args
    args.clear()
    assert_size_stride(arg0_1, (1024, 32, 1), (32, 1, 1))
    assert_size_stride(arg1_1, (1, 32, 128), (4096, 128, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0) # no-op to ensure context
        buf0 = empty_strided((1024, 32, 128), (4096, 128, 1), device='cuda',
dtype=torch.float32)
        # Source Nodes: [add], Original ATen: [aten.add]
        stream0 = get_cuda_stream(0)
```

```
        triton_poi_fused_add_0.run(arg0_1, arg1_1, buf0, 4194304,
grid=grid(4194304), 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((1024, 32, 1), (32, 1, 1), device='cuda:0',
dtype=torch.float32)
    arg1_1 = rand_strided((1, 32, 128), (4096, 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)
```