THE LINUX FOUNDATION

OPEN SOURCE SUMMIT INDIA

Speeding Up Your ML workload: torch.compile Deep Dive

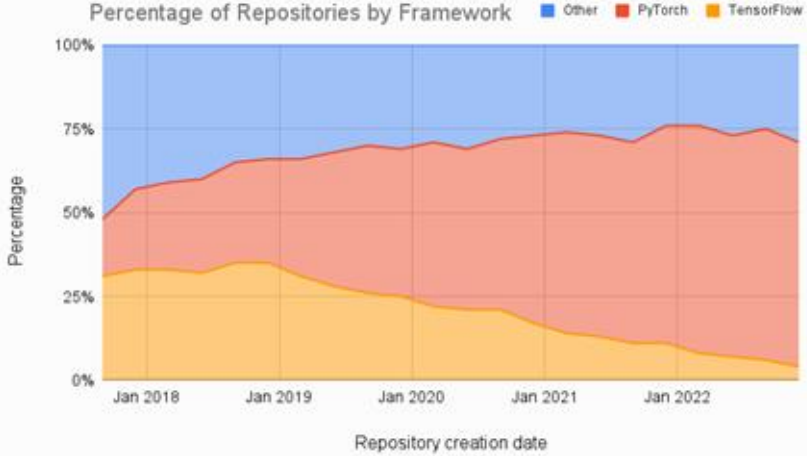
Aishwariya C, Priyanka N, Kavya G, and Chander G,
IBM Research

Contents

- Why Compile
- What is Compile
- Torch.compile Internals
- Speed Up Numbers
- IBM Spyre – Compile-first Stack
- How to Contribute

Eager Mode of Execution

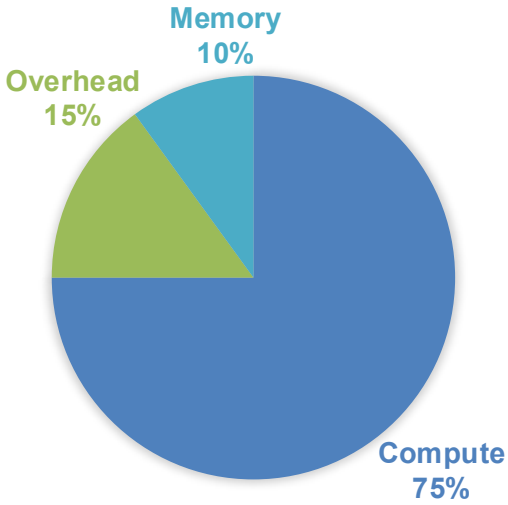
- PyTorch by default runs models in **eager** mode
  - Dynamic execution
  - Prototyping: Easy to debug, highly flexible
  - Computation graph of code is created at runtime
  - Bad performance for production



Performance Gaps

Regime	What's happening	How to spot it
Compute-bound	GPU doing math	High SM utilisation
Memory-bound	Shuffling tensors to DRAM	Low TFLOPS, high BW
Overhead-bound	Python dispatch & launches	CPU busy, GPU idle ← most models

Time breakdown for a typical transformer training step

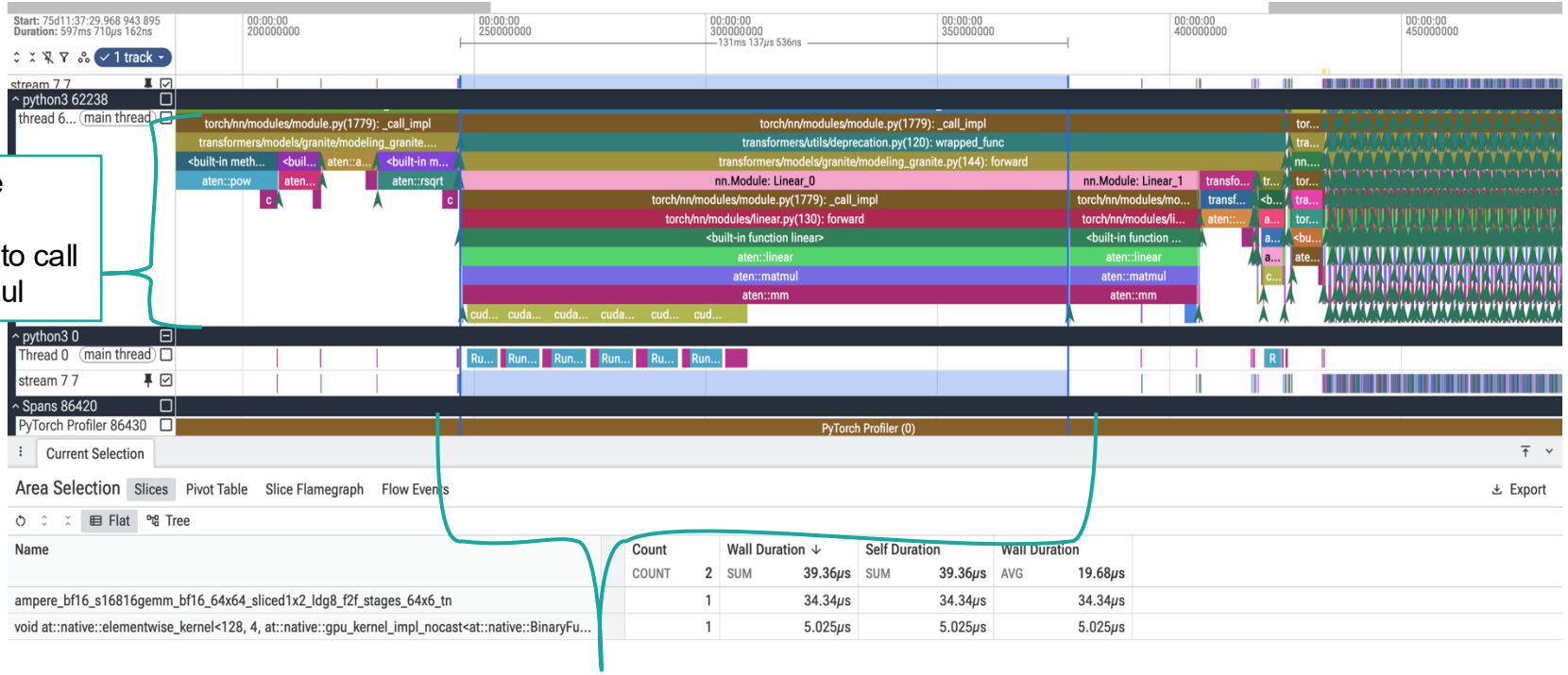


Your GPU is fast. The question is whether you are letting it run.

Let the Profiler Show You the Crime Scene


```
import torch
from torch import profiler
from transformers import AutoModelForCausalLM, AutoTokenizer

model_name = "ibm-granite/granite-3.3-8b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, dtype=torch.bfloat16).to("cuda")
inputs = tokenizer("What is the capital of France?", return_tensors="pt").to("cuda")

with torch.no_grad():
    with profiler.profile(
        activities=[profiler.ProfilerActivity.CPU, profiler.ProfilerActivity.CUDA],
        record_shapes=True,
        profile_memory=True,
        with_stack=True,
    ) as p:
        model(**inputs)

p.export_chrome_trace("trace-granite.json")
```

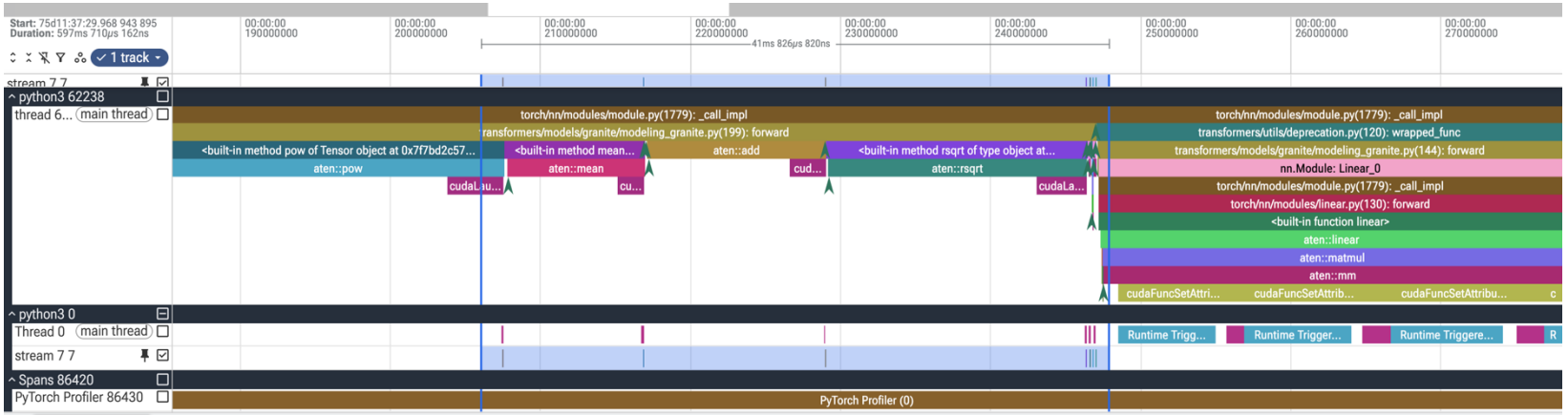
Let the Profiler Show You the Crime Scene



Multiple Python frames to call a matmul

131ms: GPU idle — CPU dispatching one Linear layer

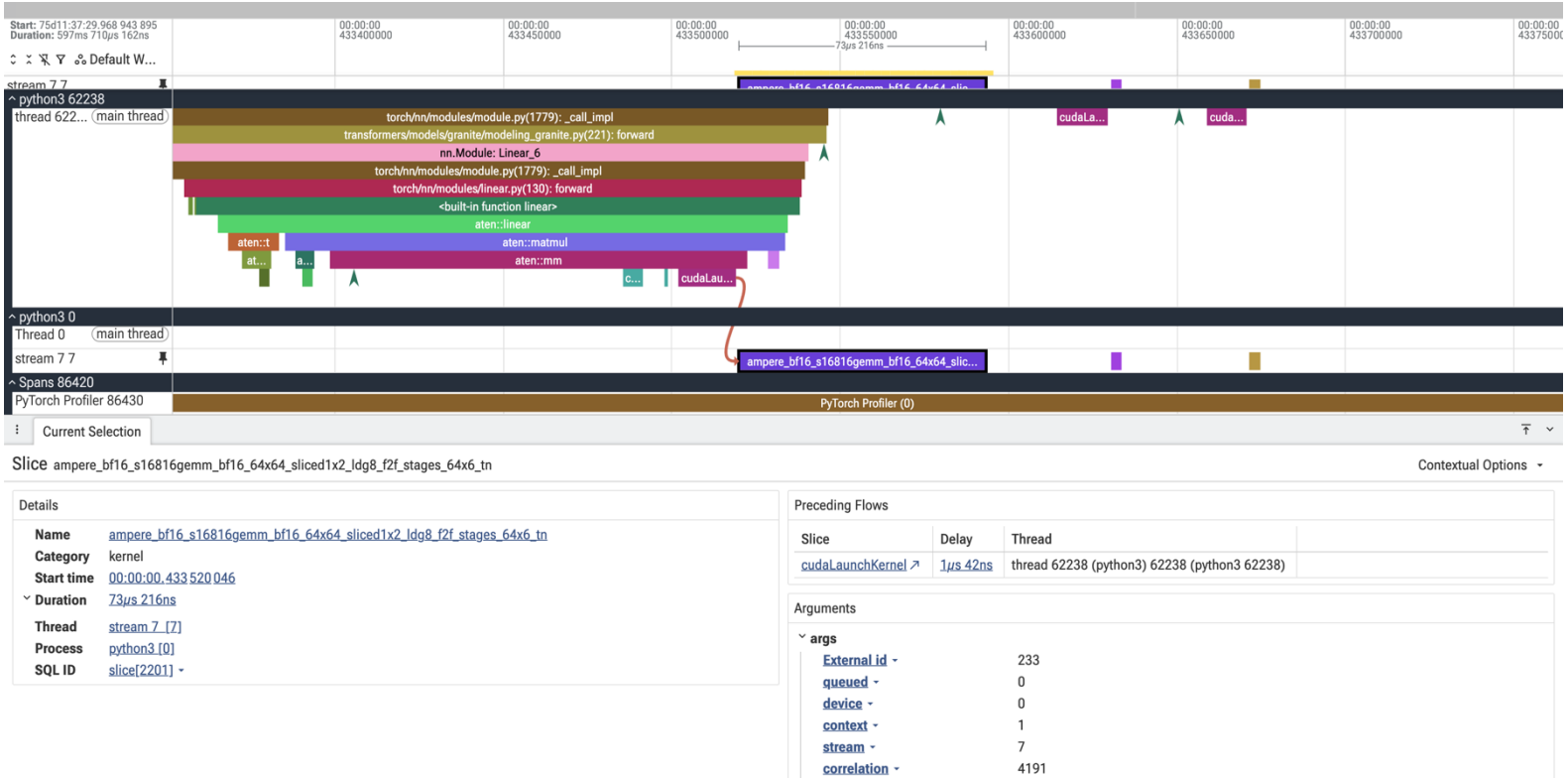
Let the Profiler Show You the Crime Scene



Name	Count COUNT 7	Wall Duration ↓ SUM 26.37µs	Self Duration SUM 26.37µs	Wall Duration AVG 3.767µs
void at::native::reduce_kernel<512, 1, at::native::ReduceOp<float, at::native::MeanOps<float, float, ...	1	7.425µs	7.425µs	7.425µs
void at::native::elementwise_kernel<128, 4, at::native::gpu_kernel_impl_nocast<at::native::BinaryFu...	1	5.025µs	5.025µs	5.025µs
void at::native::elementwise_kernel<128, 2, at::native::gpu_kernel_impl_nocast<at::native::BinaryFu...	1	4.064µs	4.064µs	4.064µs
void at::native::vectorized_elementwise_kernel<4, at::native::bfloat16_copy_kernel_cuda(at::TensorI...	1	2.560µs	2.560µs	2.560µs
void at::native::vectorized_elementwise_kernel<4, at::native::(anonymous namespace)::pow_tensor...	1	2.560µs	2.560µs	2.560µs
void at::native::vectorized_elementwise_kernel<4, at::native::rsqrt_kernel_cuda(at::TensorIteratorBa...	1	2.464µs	2.464µs	2.464µs
void at::native::vectorized_elementwise_kernel<4, at::native::CUDAFunctorOnSelf_add<float>, std::a...	1	2.272µs	2.272µs	2.272µs

7 kernels, ~2-7µs each — one DRAM round-trip per op, no fusion

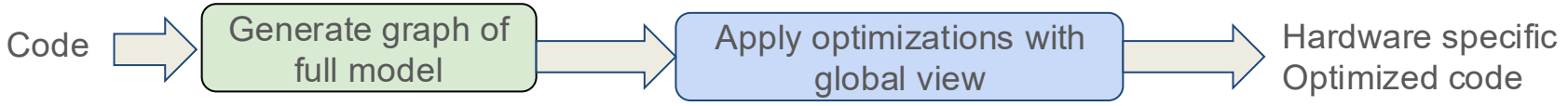
Let the Profiler Show You the Crime Scene



73µs GEMM — part of the 2.2% of time the GPU is actually computing

Graph Mode of Execution

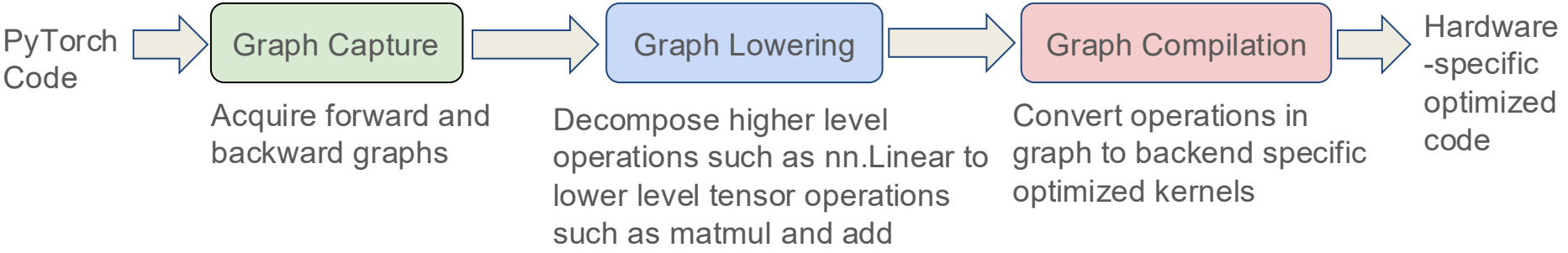
- Graph mode enables the application of optimizations such as scheduling and operator fusion for faster execution/lower overheads



Enter torch.compile

- Just-in-Time compiler introduced in PyTorch 2.0
- No framework rewrite required, works with existing PyTorch code

```
@torch.compile
def foo(x: torch.Tensor, y: torch.Tensor):
    z = x * y
    w = z + x
    return w
```



Our Results — MobileNet v2

Ran the following:

- Model: MobileNet v2
- Dataset: ImageNet with 1000 sample images

Without Compile

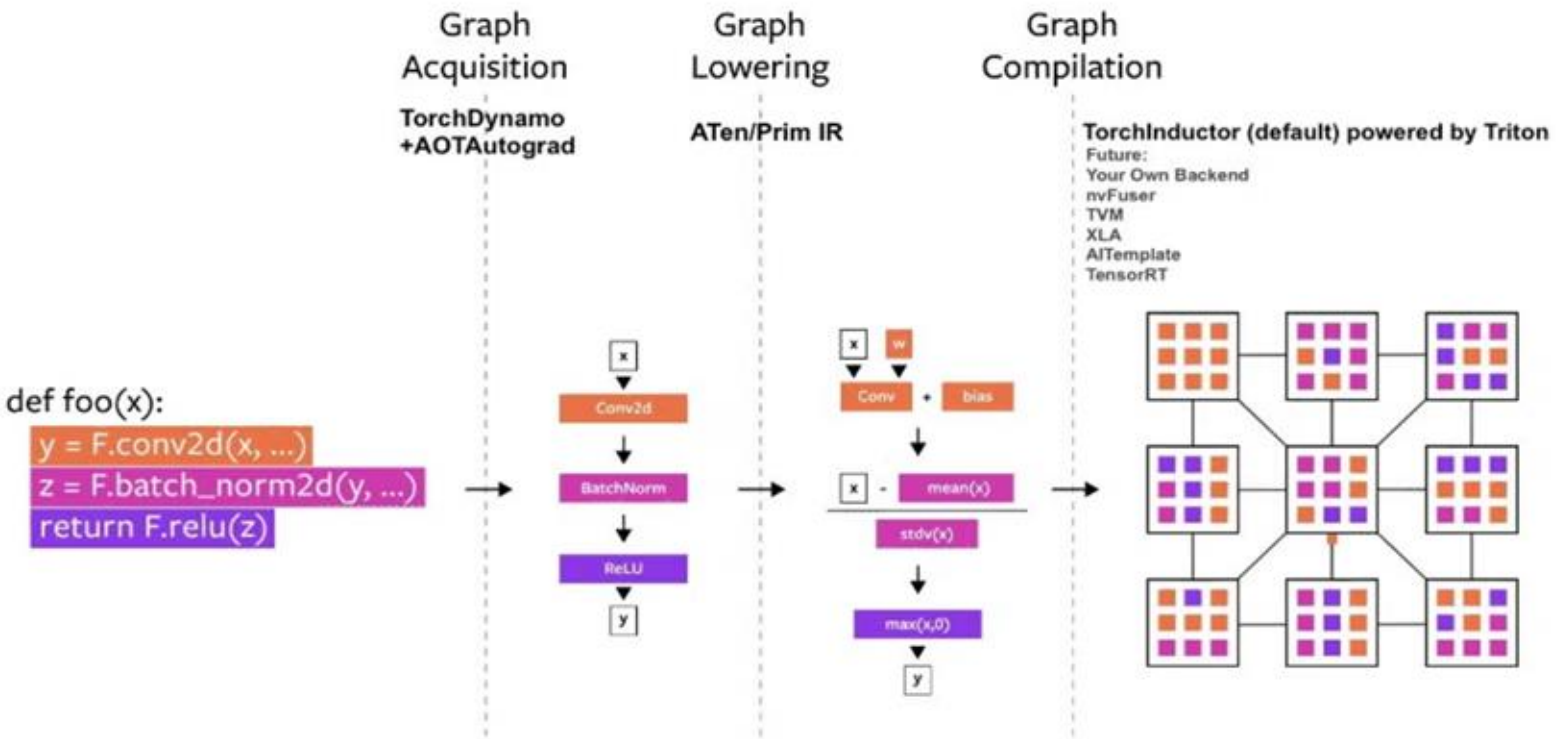
```
Batch Size: 32
Average Latency: 26.55 ± 0.28 ms
Min Latency: 26.34 ms
Max Latency: 27.74 ms
Throughput: 1205.34 samples/sec
```

With Compile

```
Batch Size: 32
Average Latency: 17.86 ± 0.17 ms
Min Latency: 17.68 ms
Max Latency: 18.33 ms
Throughput: 1791.60 samples/sec
```

32% reduction in avg latency and 48% increase in throughput

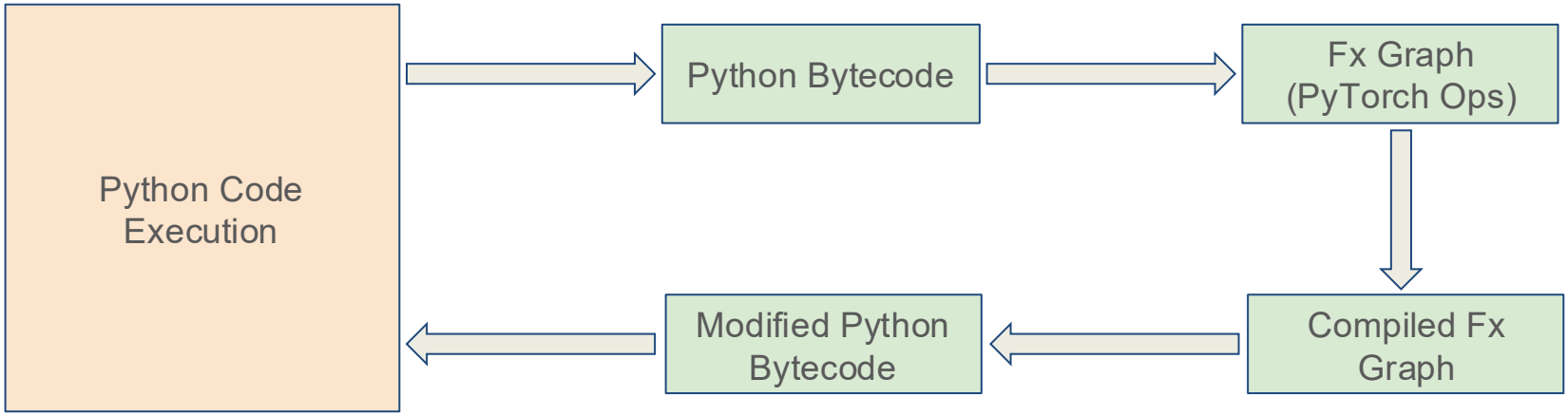
What happens under the hood



Ref: https://pytorch.org/get-started/pytorch-2-x/

Component I - Torch Dynamo

- Python level JIT compiler
  - Designed to allow graph compilation in PyTorch programs
  - Emits Fx Graphs containing sequences of PyTorch operations

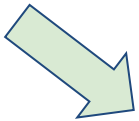


Ref: https://docs.pytorch.org/docs/stable/torch.compiler_dynamo_overview.html

Features of Torch Dynamo - Graph Breaks

- Unlike other tracers, torch dynamo does not crash when it encounters unknown code.
- It simply falls back to the CPython Interpreter to run the problematic code.

```
@torch.compile(backend=my_compiler)
def fn(x, y):
    a = torch.cos(x)
    b = torch.sin(y)
    return a + b
```



```
opcode         name    target                                                  args         kwargs
-------------  ------  ------------------------------------------------------  -----------  --------
placeholder    l_x_    L_x_                                                    ()           {}
placeholder    l_y_    L_y_                                                    ()           {}
call_function  a       <built-in method cos of type object at 0x7b565621ff00>  (l_x_,)      {}
call_function  b       <built-in method sin of type object at 0x7b565621ff00>  (l_y_,)      {}
call_function  add     <built-in function add>                                 (a, b)       {}
output         output  output                                                  ((add,),)    {}
```


Tabular view of FX graph

Features of Torch Dynamo - Graph Breaks


```python
@torch.compile(backend=my_compiler)
def fn(x, y):
    a = torch.cos(x)
    b = torch.sin(y)
    print("Hello")
    return a + b
```

```python
explanation = dynamo.explain(fn)(torch.randn(10), torch.randn(10))
print(explanation)
```

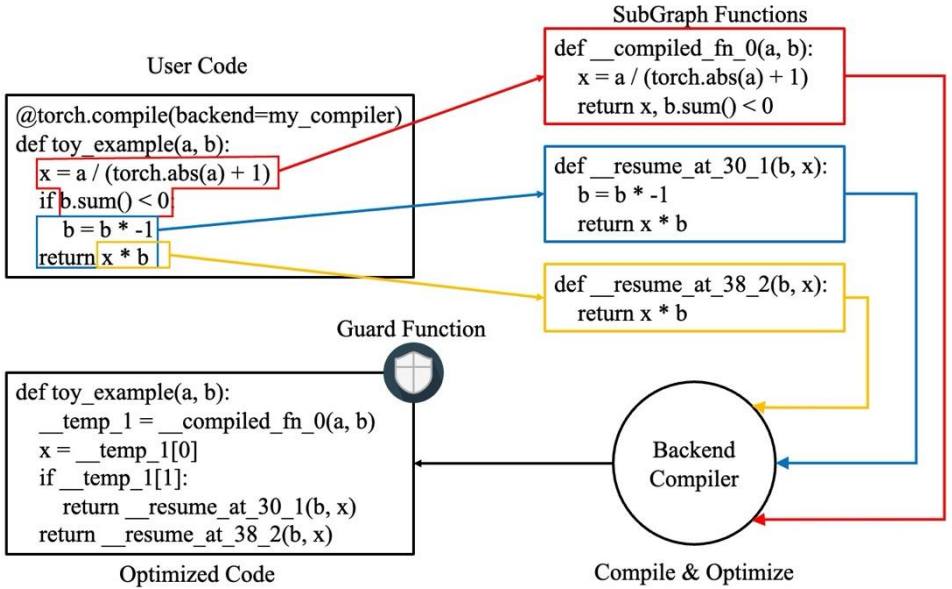
```
Hello
Graph Count: 2
Graph Break Count: 1
Op Count: 3
Break Reasons:
  Break Reason 1:
    Reason: builtin: print [<class 'torch._dynamo.variables.constant.ConstantVariable'>] False
    User Stack:
      <FrameSummary file /tmp/ipython-input-10-3243079242.py, line 12 in fn>
```

```
opcode         name    target                                                  args       kwargs
-------------  ------  ------------------------------------------------------  ---------  --------
placeholder    l_x_    L_x_                                                    ()         {}
placeholder    l_y_    L_y_                                                    ()         {}
call_function  a       <built-in method cos of type object at 0x7b565621ff00>  (l_x_,)    {}
call_function  b       <built-in method sin of type object at 0x7b565621ff00>  (l_y_,)    {}
output         output  output                                                  ((a, b),)  {}
Hello
opcode         name    target                   args          kwargs
-------------  ------  -----------------------  ------------  --------
placeholder    l_a_    L_a_                     ()            {}
placeholder    l_b_    L_b_                     ()            {}
call_function  add     <built-in function add>  (l_a_, l_b_)  {}
output         output  output                   ((add,),)     {}
```

Features of Torch Dynamo - Guards

Guards specify conditions (e.g., devices, data types, shapes, values) under which functions can be optimized

- If guard conditions satisfied, Dynamo replaces functions by their optimized versions in Python bytecode
- Otherwise, recompilation takes place under the new guard.



Ref: https://docs.pytorch.org/docs/2.12/user_guide/torch_compiler/torch.compiler_dynamo_overview.html

Features of Torch Dynamo - Guards

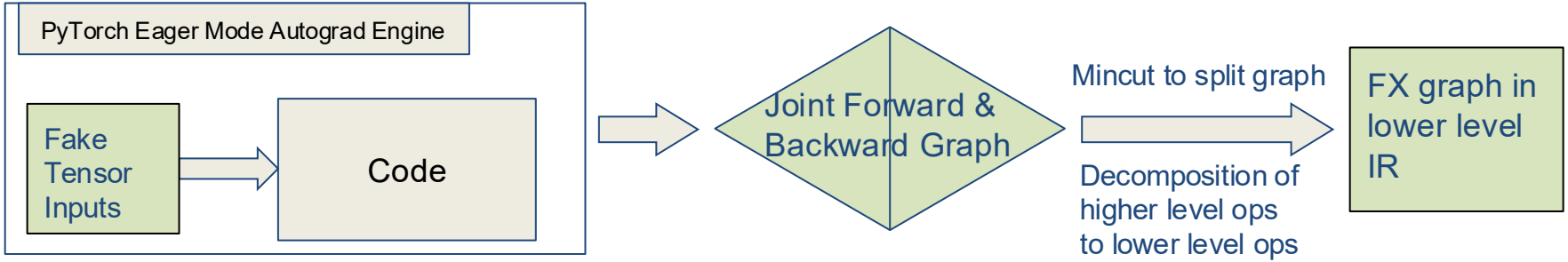

```python
import torch

@torch.compile
def fn(x, n):
    y = x ** 2
    if n >= 0:
        return (n + 1) * y
    else:
        return y / n

x = torch.randn(200)
fn(x, 2)
fn(x, 3)
fn(x, -2)
```

```
uts.py
[__guards] GUARDS:
[__guards]
[__guards] TREE_GUARD_MANAGER:
[__guards] +- RootGuardManager
[__guards] | +- DEFAULT_DEVICE: utils_device.CURRENT_DEVICE == None                          # _dynamo/output_graph.py:493 in init_ambient_guards
[__guards] | +- GLOBAL_STATE: ___check_global_state()
[__guards] | +- TORCH_FUNCTION_MODE_STACK: ___check_torch_function_mode_stack()
[__guards] | +- GuardManager: source=L['n'], accessed_by=DictGetItemGuardAccessor('n')
[__guards] | | +- EQUALS_MATCH: L['n'] == 2                                                 # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn
[__guards] | +- GuardManager: source=L['x'], accessed_by=DictGetItemGuardAccessor('x')
[__guards] | | +- TENSOR_MATCH: check_tensor(L['x'], Tensor, DispatchKeySet(CPU, BackendSelect, ADInplaceOrView, AutogradCPU), torch.float32, device=None, re
uts.py:5 in fn
[__guards] | | +- NO_HASATTR: hasattr(L['x'], '_dynamo_dynamic_indices') == False           # y = x ** 2  # sers/aishwariya/tuts.py:5 in fn
[__guards]
[__guards] Guard eval latency = 0.26 us
[__recompiles] Recompiling function fn in /Users/aishwariya/tuts.py:3
[__recompiles]     triggered by the following guard failure(s):
[__recompiles]     - 0/0: L['n'] == 2                                          # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn
[__guards] GUARDS:
[__guards]
[__guards] TREE_GUARD_MANAGER:
[__guards] +- RootGuardManager
[__guards] | +- DEFAULT_DEVICE: utils_device.CURRENT_DEVICE == None                          # _dynamo/output_graph.py:493 in init_ambient_guards
[__guards] | +- GLOBAL_STATE: ___check_global_state()
[__guards] | +- TORCH_FUNCTION_MODE_STACK: ___check_torch_function_mode_stack()
[__guards] | +- GuardManager: source=L['n'], accessed_by=DictGetItemGuardAccessor('n')
[__guards] | | +- TYPE_MATCH: ___check_type_id(L['n'], 4361081088)                          # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn
[__guards] | +- GuardManager: source=L['x'], accessed_by=DictGetItemGuardAccessor('x')
[__guards] | | +- TENSOR_MATCH: check_tensor(L['x'], Tensor, DispatchKeySet(CPU, BackendSelect, ADInplaceOrView, AutogradCPU), torch.float32, device=None, re
uts.py:5 in fn
[__guards] | | +- NO_HASATTR: hasattr(L['x'], '_dynamo_dynamic_indices') == False           # y = x ** 2  # sers/aishwariya/tuts.py:5 in fn
[__guards] +- LAMBDA_GUARD: 0 <= L['n']  # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn (_dynamo/variables/tensor.py:1201 in evaluate_expr)
[__guards]
[__guards] Guard eval latency = 0.28 us
[__recompiles] Recompiling function fn in /Users/aishwariya/tuts.py:3
[__recompiles]     triggered by the following guard failure(s):
[__recompiles]     - 0/1: 0 <= L['n']  # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn (_dynamo/variables/tensor.py:1201 in evaluate_expr)
[__recompiles]     - 0/0: L['n'] == 2                                          # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn
[__guards] GUARDS:
[__guards]
[__guards] TREE_GUARD_MANAGER:
[__guards] +- RootGuardManager
[__guards] | +- DEFAULT_DEVICE: utils_device.CURRENT_DEVICE == None                          # _dynamo/output_graph.py:493 in init_ambient_guards
[__guards] | +- GLOBAL_STATE: ___check_global_state()
[__guards] | +- TORCH_FUNCTION_MODE_STACK: ___check_torch_function_mode_stack()
[__guards] | +- GuardManager: source=L['n'], accessed_by=DictGetItemGuardAccessor('n')
[__guards] | | +- TYPE_MATCH: ___check_type_id(L['n'], 4361081088)                          # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn
[__guards] | +- GuardManager: source=L['x'], accessed_by=DictGetItemGuardAccessor('x')
[__guards] | | +- TENSOR_MATCH: check_tensor(L['x'], Tensor, DispatchKeySet(CPU, BackendSelect, ADInplaceOrView, AutogradCPU), torch.float32, device=None, re
uts.py:5 in fn
[__guards] | | +- NO_HASATTR: hasattr(L['x'], '_dynamo_dynamic_indices') == False           # y = x ** 2  # sers/aishwariya/tuts.py:5 in fn
[__guards] +- LAMBDA_GUARD: L['n'] <= -1  # if n >= 0:  # sers/aishwariya/tuts.py:6 in fn (_dynamo/variables/tensor.py:1201 in evaluate_expr)
[__guards]
[__guards] Guard eval latency = 0.28 us
```

Recompilation upon guard failure

Component II - AOT Autograd

- Captures both forward and backward computations as FX graphs ahead of time
- It decides what values to save in order to run the forward and backward passes with minimal memory overheads and memory access latency



Component III - Inductor

- Default compiler backend for TorchDynamo
- Translates PyTorch programs into Triton for GPU and C++/OpenMP for CPU
- Performs different optimizations such as:

```
tmp1 = x + y
tmp2 = tmp1 * z
out  = relu(tmp2)

# Can be reduced to
out = relu((x + y)* z)
```

Operator fusion

```
y = sin(x)
z = sin(x)
return z

# Can be reduced to
z = sin(x)
return z
```

Dead code eliminiation

```
scale = 2.0 * 4.0
y = x * scale

# Can be reduced to
y = x * 8.0
```

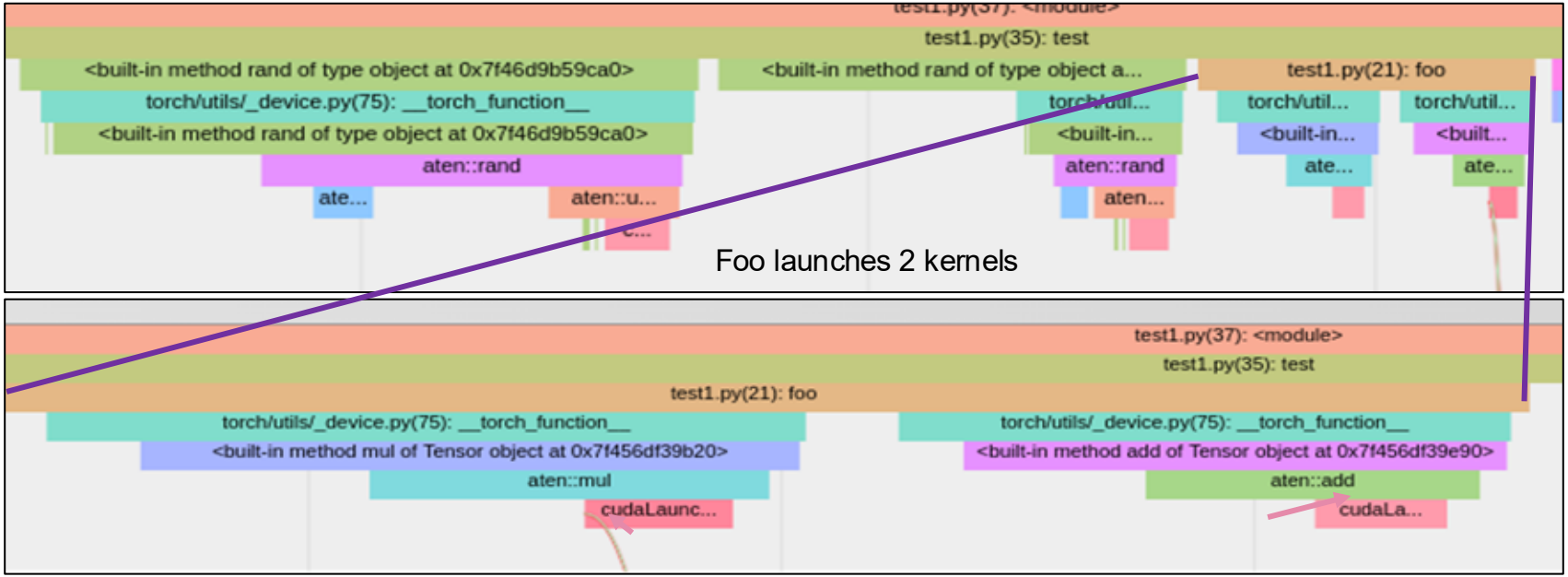
Constant folding

```
tmp = x + 1.0
y = tmp * 2.0

# Can be reduced to
y = x + 1.0
y *= 2.0
```

Buffer Reuse

Example of operator fusion

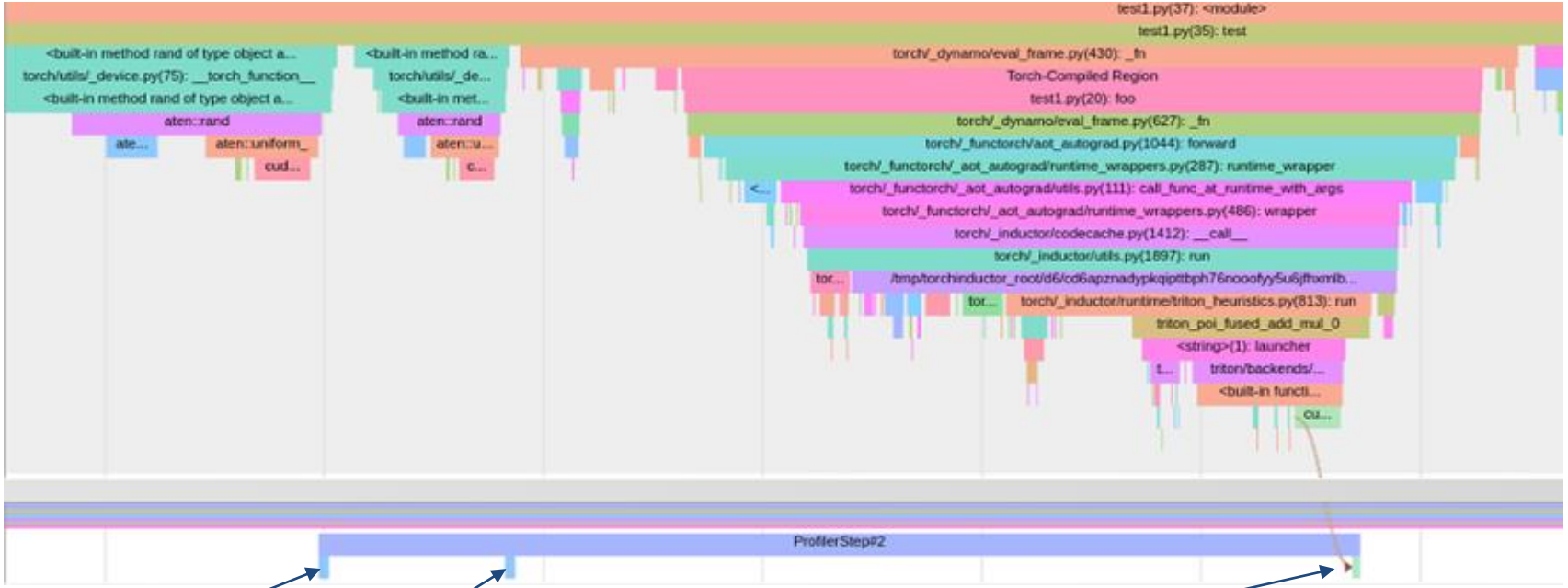
```
def foo(x: torch.Tensor, y: torch.Tensor):
    z = x * y
    w = z + x
    return w
```



CPU Profile graph without torch.compile

Example of operator fusion


```
@torch.compile
def foo(x: torch.Tensor, y: torch.Tensor):
    z = x * y
    w = z + x
    return w
```



Fused Kernel in Triton


```
triton_poi_fused_add_mul_0 = async_compile.triton('triton_', '''
…
@triton.jit
def triton_(in_ptr0, in_ptr1, out_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 25
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x0 = xindex
    tmp0 = tl.load(in_ptr0 + (x0), xmask)
    tmp1 = tl.load(in_ptr1 + (x0), xmask)
    tmp2 = tmp0 * tmp1
    tmp3 = tmp2 + tmp0
    tl.store(out_ptr0 + (x0), tmp3, xmask)
…


stream0 = get_raw_stream(0)
triton_poi_fused_add_mul_0.run(arg0_1, arg1_1, buf0, 25, grid=grid(25), stream=stream0)
```

TODO: Try to generate the torch profiler trace for granite in compile mode and try to find out the differences.

Speed Up Numbers – Broader Benchmarks

TorchBench CPU (green=0 breaks · yellow=few · red=many)

Model	Speedup	Graph breaks
Background_Matting	1.51×	0
llama	1.36×	0
basic_gnn_edgecnn	1.28×	0
BERT_pytorch	1.25×	0
basic_gnn_gcn	1.11×	6
alexnet	1.05×	0
basic_gnn_gin	1.04×	0
detectron2_fcos_r_50_fpn	1.06×	20
detectron2_fasterrcnn_r_50_dc5	1.05×	32
detectron2_fasterrcnn_r_50_fpn	1.03×	34
detectron2_fasterrcnn_r_101_dc5	1.03×	32
detectron2_fasterrcnn_r_101_fpn	1.02×	34
detectron2_maskrcnn_r_101_c4	1.02×	44
detectron2_fasterrcnn_r_101_c4	0.97×	32
detectron2_fasterrcnn_r_50_c4	0.96×	32

Published — PyTorch 2.x TorchBench (GPU)

- ~1.3× geomean speedup across 163 open-source models
- BERT inference on A100: up to 2×
- LLaMA inference on A100: up to 1.8×

Ref: pytorch.org/get-started/pytorch-2.0

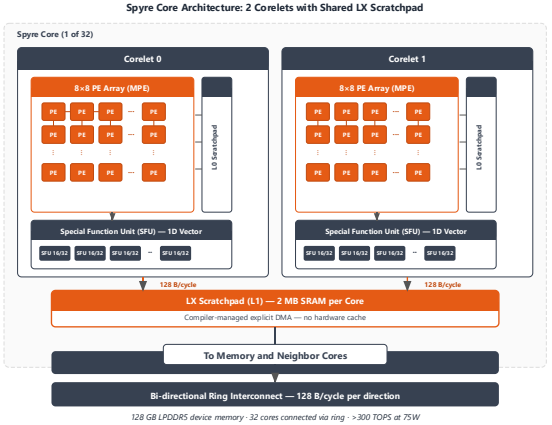
Note: Detectron2 (32–44 breaks) still gets speedups.

IBM Spyre – Example of Compile-first Stack

IBM's dataflow accelerator — built compile-first on torch.compile

The hardware

- 32 AI cores — mixed-precision SIMD-systolic arrays
- Cores are ring-connected — data flows explicitly between them
- No hardware cache — explicitly managed on-chip scratchpad memory
- **Implication: the compiler must own all data movement decisions**



Three inductor extensions (all out-of-tree)

- Tile-based tensor layouts — handles block-structured data movement
- Multi-core work-division — partitions tiles across 32 cores at compile time
- Scratchpad scheduling — compiler orchestrates on-chip memory explicitly

Ref: research.ibm.com/blog/pytorch-support-ibm-spyre

Your Entry Point to torch.compile

01
File bugs & graph break reports

Anyone can do this

- Run your model: model = torch.compile(model)
- Enable logging: torch._dynamo.config.verbose = True
- Real-world graph breaks drive the roadmap

02
Improve TorchInductor

Python / Triton experience required

- Inductor is mostly Python: torch/_inductor/ in the pytorch repo
- Entry points: fusion passes, layout optimisation, op lowerings
- Good first issues tagged module: inductor on GitHub

03
Build a new hardware backend

Systems / compiler experience required

- Register via torch._dynamo.register_backend
- Implement an inductor lowering pass for your target
- Reference: cudagraphs/onnx

Tutorial: docs.pytorch.org/tutorials/intermediate/torch_compile_tutorial.html
Compiler docs: docs.pytorch.org/docs/stable/user_guide/torch_compiler/
Dev forum: dev-discuss.pytorch.org

What We Didn't Cover: Distributed Training

- Data Parallelism (DDP)
- Tensor Parallelism (TP)
- Pipeline Parallelism (PP)
- Full Sharded Data Parallelism (FSDP)
  - With variants - like Full/Hybrid Shard
- Communication collectives – NCCL
- Toy Simulator - https://github.ibm.com/chandergovind/dist-train-simulator

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT INDIA

Utilities - Modes, Configs

torch.compile() Arguments:

- modes:
  - `default`: efficient compilation with no extra memory and low compilation time
  - `reduce-overhead`: maximum framework overhead reduction, with little extra memory
  - `max-autotune`: fastest possible code, but long compilation time
- `fullgraph=True` → to create single graph with no graph break
- `dynamic=True` → all integers and shapes marked as dynamic

Other utilities:

- `torch._dynamo.utils.compile_times()` → to display time spent in each compilation phase
- `torch._dynamo.config.recompile_limit` → to limit the number of recompilations
- `torch._dynamo.explain()` → to display number of graph breaks and their causes
- `torch._dynamo.reset()` → to reset all artifacts created by dynamo
- `export TORCH_LOGS="graph_code,graph_sizes,dynamo,guards,output_code,graph_breaks"`

Deep Dive

1. https://docs.pytorch.org/docs/2.12/user_guide/torch_compiler/torch.compiler.html
2. https://docs.google.com/document/d/1y5CRfMLdwEoF1nTk9q8qEu1mgMUuUtvhklPKJ2emLU8/edit?tab=t.0#heading=h.ivdr7fmrbeab
3. https://colab.research.google.com/drive/1Zh-Uo3TcTH8yYJF-LLo5rjlHVMtqvMdf?usp=sharing#scrollTo=ObOktQOeko5h
4. https://docs.pytorch.org/docs/stable/torch.compiler_dynamo_deepdive.html
5. https://docs.pytorch.org/assets/pytorch2-2.pdf
6. https://www.youtube.com/watch?v=rn-kJQ-7JmQ
7. https://www.youtube.com/watch?v=ppWKVg-VxmQ
8. https://www.youtube.com/watch?v=GmhnYe9QQoM
9. https://www.youtube.com/watch?v=5FNHwPIyHr8
10. https://docs.google.com/document/d/1GgvOe7C8_NVOMLOCwDaYV1mXXyHMXY7ExoewHqooxrs/edit?tab=t.0#heading=h.fh8zzonyw8ng
11. https://medium.com/data-science/how-pytorch-2-0-accelerates-deep-learning-with-operator-fusion-and-cpu-gpu-code-generation-35132a85bd26