SIG OpenXLA Community Meeting

January 24, 2023

OpenXLA

What is OpenXLA?

Open, state-of-art ML compiler ecosystem, built collaboratively with Hardware & Software partners, using the best of XLA & MLIR.

Introductions

Welcome!

- Welcome to any new attendees? What are you looking to focus on?
- SIG member organizations:
    - Alibaba
    - AMD
    - Apple
    - ARM
    - AWS
    - Google
    - Intel
    - Meta
    - NVIDIA

SIG Collaboration

Reference material for our new collaborators

Our Meetings

- Monthly on Zoom, 3rd Tuesday @ 8AM PT (except this week)
- Rotating meeting host & scribe
- Proposed agenda shared by host week prior in GitHub Discussions
- Meeting minutes & slides shared publicly in the meetings archive the day after
- Meetings should include:
    - Development updates
    - Design proposals
    - Community topics

Our Collaboration Channels

Channel	Content	Access	Archive
GitHub organization	Code, Design proposals, PRs, Issues, Roadmaps	Public	N/A
Community repository	Governance, Meetings, Code of conduct	Public	Public
Community discussions	Meta discussions on openxla/community repo	Public	Public
Technical discussions	Technical discussions on individual repos: xla, stablehlo	Public	Public
Discord	Sync discussions	Open invites	Archived chats
Community meetings	Monthly live meetings	Public	Public agenda, slides, meeting minutes

OpenXLA

Development Updates

StableHLO Specification

Sandeep Dasgupta, Google
sdasgup@google.com

OpenXLA

Specification of StableHLO programming language

c

StableHLO: A stable version of HLO, which extends it with support for versioning and compatibility.

c

Motivation

Improves interoperability between ML frameworks (TF, JAX, PyTorch) and ML compilers (XLA, IREE and others)

Goals

- Reference for StableHLO producers
- SoT for various StableHLO consumers

Non Goals (in the initial version)

- Re-invent/Customize HLO/MHLO
- Automate generation of spec

XLA Operation Semantics: Features

- Significant opset coverage
- Detailed input/output types and semantics
- Assistive diagrams for non-trivial op's semantics
- Occasionally spells out ML use-case scenarios

result of applying a `ReduceWindow` operation on the `operand` array. `SelectAndScatter` can be used to backpropagate the gradient values for a pooling layer in a neural network.

picture courtesy: xla/operation_semantics

mhlo Dialect: Features

- Auto-generated from the MHLO dialect ODS, which is source of truth
- Verification constraints specified using
  - `TypeConstraints`, `AttrConstraints`, `Traits` etc., in ODS
  - Custom `op::verify` methods, in c++ implementations
- Op specifications follow same format

Learnings from HLO/MHLO specs

- Get inspired from HLO/MHLO specifications
  - Should follow a uniform structure
  - Should leverage existing verification constraints
  - Should provide assistive diagrams/examples for clarity

- Opportunities to improve existing specifications w.r.t
  - Resolve occasional ambiguities
  - Comprehensively specify constraints
  - Introduce formalism

Approach

- Bootstrap from MHLO opset
- Consult with the following
  - HLO verifier: For constraints involving inputs/outputs
  - HLO shape inference: For further constraints
  - HLO evaluator: For execution semantics
  - HloInstruction::CreateFromProto: For op metadata coverage
  - HLO developers: For resolving ambiguities

Features

Specification Features: Formal

- Syntax defined formally using EBNF grammar.
- Semantics defined semi-formally*

Benefits

- Unambiguous
- Tractable
- Foundational

* Refer to [#484](#484) in formalizing the semantics

Example (ConcatenateOp)

HLO ([full spec](#))

Concatenate composes an array from multiple array operands. The array is of the same rank as each of the input array operands (which must be of the same rank as each other) and contains the arguments in the order that they were specified.

With the exception of `dimension` all dimensions must be the same. This is because XLA does not support "ragged" arrays. Also note that rank-0 values cannot be concatenated (as it's impossible to name the dimension along which the concatenation occurs).

Example (ConcatenateOp)

HLO (full spec)

Concatenate composes an array from multiple array operands. The array is of the same rank as each of the input array operands (which must be of the same rank as each other) and contains the arguments in the order that they were specified.

With the exception of `dimension` all dimensions must be the same. This is because XLA does not support "ragged" arrays. Also note that rank-0 values cannot be concatenated (as it's impossible to name the dimension along which the concatenation occurs).

StableHLO (full spec)

Concatenates a variadic number of tensors in `inputs` along `dimension` dimension in the same order as the given arguments and produces a `result` tensor. More formally, `result[i0, ..., id, ..., iR-1] = inputs[k][i0, ..., kd, ..., iR-1]`, where:

1. `id = d0 + ... + dk-1 + kd`.
2. `d` is equal to `dimension`, and `d0`, ... are `d` th dimension sizes of `inputs`.

Example (RemainderOp)

HLO (full spec)

When `Op` is `Rem`, the sign of the result is taken from the dividend, and the absolute value of the result is always less than the divisor's absolute value.

std::fmod or std::remainder?

Example (RemainderOp)

HLO ([full spec]())

When `Op` is `Rem`, the sign of the result is taken from the dividend, and the absolute value of the result is always less than the divisor's absolute value.

std::fmod or std::remainder?

StableHLO ([full spec]())

Performs element-wise remainder of dividend `lhs` and divisor `rhs` tensors and produces a `result` tensor.

More formally, the sign of the result is taken from the dividend, and the absolute value of the result is always less than the divisor's absolute value. The remainder is calculated as `lhs - d * rhs`, where `d` is given by:

- For integers: `stablehlo.divide(lhs, rhs)`.
- For floats: `division(lhs, rhs)` from IEEE-754 with rounding attribute `roundTowardZero`.
- For complex numbers: TBD

For floating-point element types, this operation is in contrast with the `remainder` operation from IEEE-754 specification where `d` is an integral value nearest to the exact value of `lhs/rhs` with ties to even.

Example (RemainderOp)

HLO (full spec)

When `Op` is `Rem`, the sign of the result is taken from the dividend, and the absolute value of the result is always less than the divisor's absolute value.

std::fmod or std::remainder?

StableHLO (full spec)

Performs element-wise remainder of dividend `lhs` and divisor `rhs` tensors and produces a `result` tensor.

More formally, the sign of the result is taken from the dividend, and the absolute value of the result is always less than the divisor's absolute value. The remainder is calculated as `lhs - d * rhs`, where `d` is given by:

- For integers: `stablehlo.divide(lhs, rhs)`.
- For floats: `division(lhs, rhs)` from IEEE-754 with rounding attribute `roundTowardZero`.
- For complex numbers: TBD

For floating-point element types, this operation is in contrast with the `remainder` operation from IEEE-754 specification where `d` is an integral value nearest to the exact value of `lhs/rhs` with ties to even.

Specification Features: Structured

Uniform format for specification

- **Semantics**: Describes an “actionable” semantics
- **Inputs**: Name and type of the input operands
- **Outputs:** Name and type of output operands
- **Constraints:** Constraints on inputs and outputs
- **Example:** An example usage

Example (CeilOp)

ceil

Semantics

Performs element-wise ceil of `operand` tensor and produces a `result` tensor. Implements the `roundToIntegralTowardPositive` operation from the IEEE-754 specification.

Inputs

Name	Type	Constraints
`operand`	tensor of floating-point type	(C1)

Outputs

Name	Type	Constraints
`result`	tensor of floating-point type	(C1)

Constraints

- (C1) `operand` and `result` have the same type.

Examples

```
// %operand: [-0.8166, -0.2530, 0.2530, 0.8166, 2.0]
%result = "stablehlo.ceil"(%operand) : (tensor<5xf32>) -> tensor<5xf32>
// %result: [-0.0, -0.0, 1.0, 1.0, 2.0]
```

picture courtesy: stablehlo/docs/spec.md

Specification Features: Constraints

- **Verification constraints:** Involving inputs/outputs
- **Type inference constraints:** On type(s) of output(s)

Example (ReduceWindowOp)

Verification constraints

Constraints

- (C1) size(`inputs`) = size(`init_values`) = size(`results`) = N and N $\geq$ 1.
- (C2) All `inputs` have the same shape.
- (C3) `element_type(inputs[k]) = element_type(init_values[k])` for any k $\in$ [0, N).
- (C4) size(`window_dimensions`) = rank(`inputs[0]`).
- (C5) `window_dimensions[i]` > 0 for all i $\in$ [0, size(`window_dimensions`)).
- (C6) size(`window_strides`) = rank(`inputs[0]`).
- (C7) `window_strides[i]` > 0 for all i $\in$ [0, size(`window_strides`)).
- (C8) size(`base_dilations`) = rank(`inputs[0]`).
- (C9) `base_dilations[i]` > 0 for all i $\in$ [0, size(`base_dilations`)).
- (C10) size(`window_dilations`) = rank(`inputs[0]`).
- (C11) `window_dilations[i]` > 0 for all i $\in$ [0, size(`window_dilations`)).
- (C12) dim(`padding` , 0) = rank(`inputs[0]`) and dim(`padding` , 1) = 2.
- (C13) `body` has type `(tensor<E0>, ..., tensor<EN-1>, tensor<E0>, ..., tensor<EN-1>) -> (tensor<E0>, ..., tensor<EN-1>)` where `Ek = element_type(inputs[0])`.

Type inference constraints

- (C14) All `results` have the same shape.
- (C15) `shape(results[0]) = num_windows`
    - `dilated_input_shape = shape(inputs[0]) == 0 ? 0 : (shape(inputs[0]) - 1) * base_dilations + 1`.
    - `padded_input_shape = padding[:, 0] + dilated_input_shape + padding[:, 1]`.
    - `dilated_window_shape = window_dimensions == 0 ? 0 : (window_dimensions - 1) * window_dilations + 1`.
    - `num_windows = (padded_input_shape == 0 || dilated_window_shape > padded_input_shape) ? 0 : floor((padded_input_shape - dilated_window_shape) / window_strides) + 1`.
- (C16) `element_type(results[k]) = element_type(init_values[k])` for any k $\in$ [0, N).

Future work

Improving Quality & Coverage of Spec

- Cover full spectrum (Dynamism, Quantization, Modularity)
  - Number of Ops already specced: 79% [94/118]
- Formalising Syntax and Semantics github-issue
  - Facilitate introducing dynamism, quantization with minimal changes to spec

Reference Implementation

“Readable” reference implementation of the specification

Conformance Suite

- A test suite that consumers can plug into and run on their hardware
- Started talking to various groups about this. Please let us know if you have thoughts about this! github-issue

Contribute Back!

Contributing to XLA Operation Semantics doc.

Takeaways

- Successfully specced statically-shaped StableHLO
- Stay tuned for
  - The opset to be augmented with Dynamism, Quantization.
  - A reference implementation of the opset.

Q&A

AWS Trainium and PyTorch/XLA

Amith Mamidala, Amazon Web Services

amithrm@amazon.com

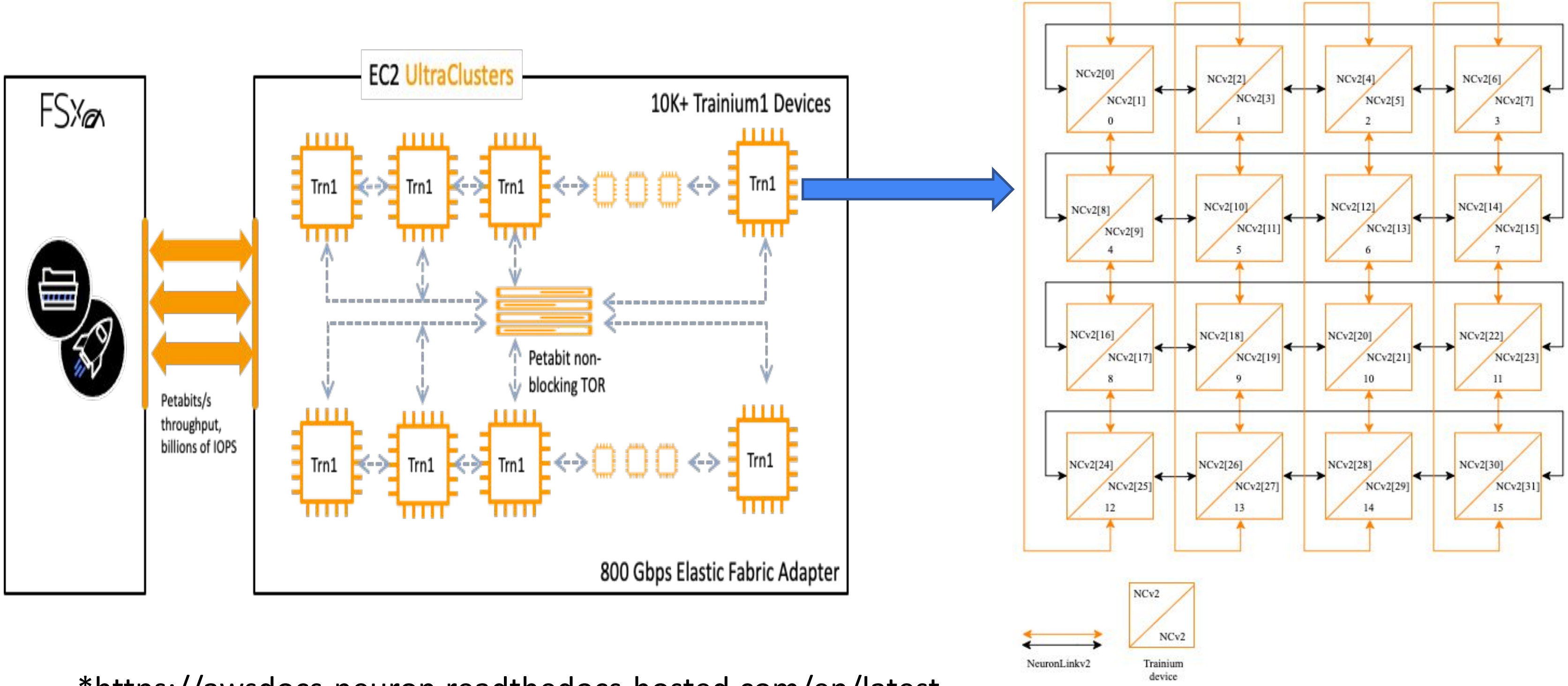
OpenXLA

Outline

- Motivation & High level Lowering flow
- Execution Flow: XLA aware orchestration
- Experience optimizing GPT3, Bert with XLA
    - Mismatches/Gaps encountered
- Future plans

Trainium Architecture

Each Trainium Device (Two Neuron Core)

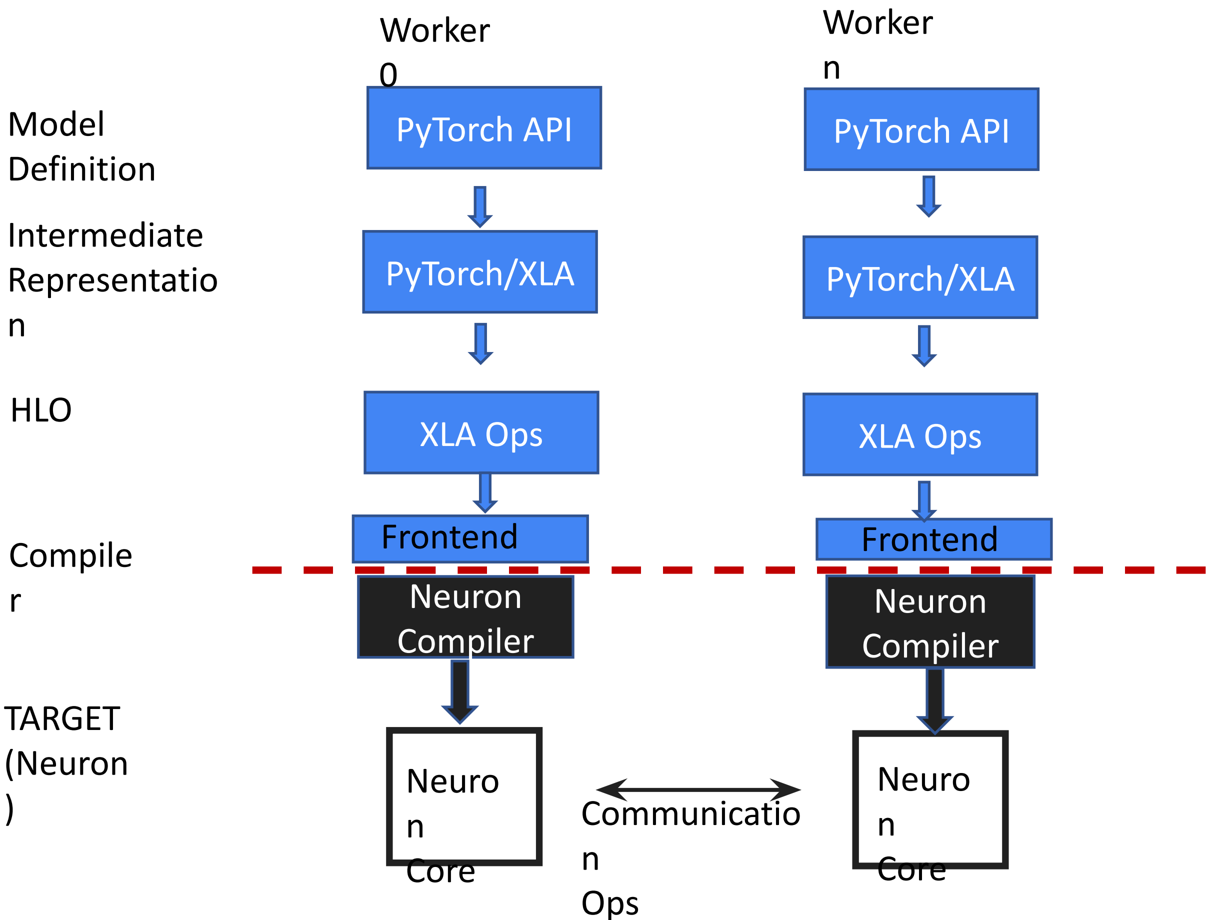


*https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/arch/neuron-hardware/trn1-arch.html#aws-trn1-arch

Goals

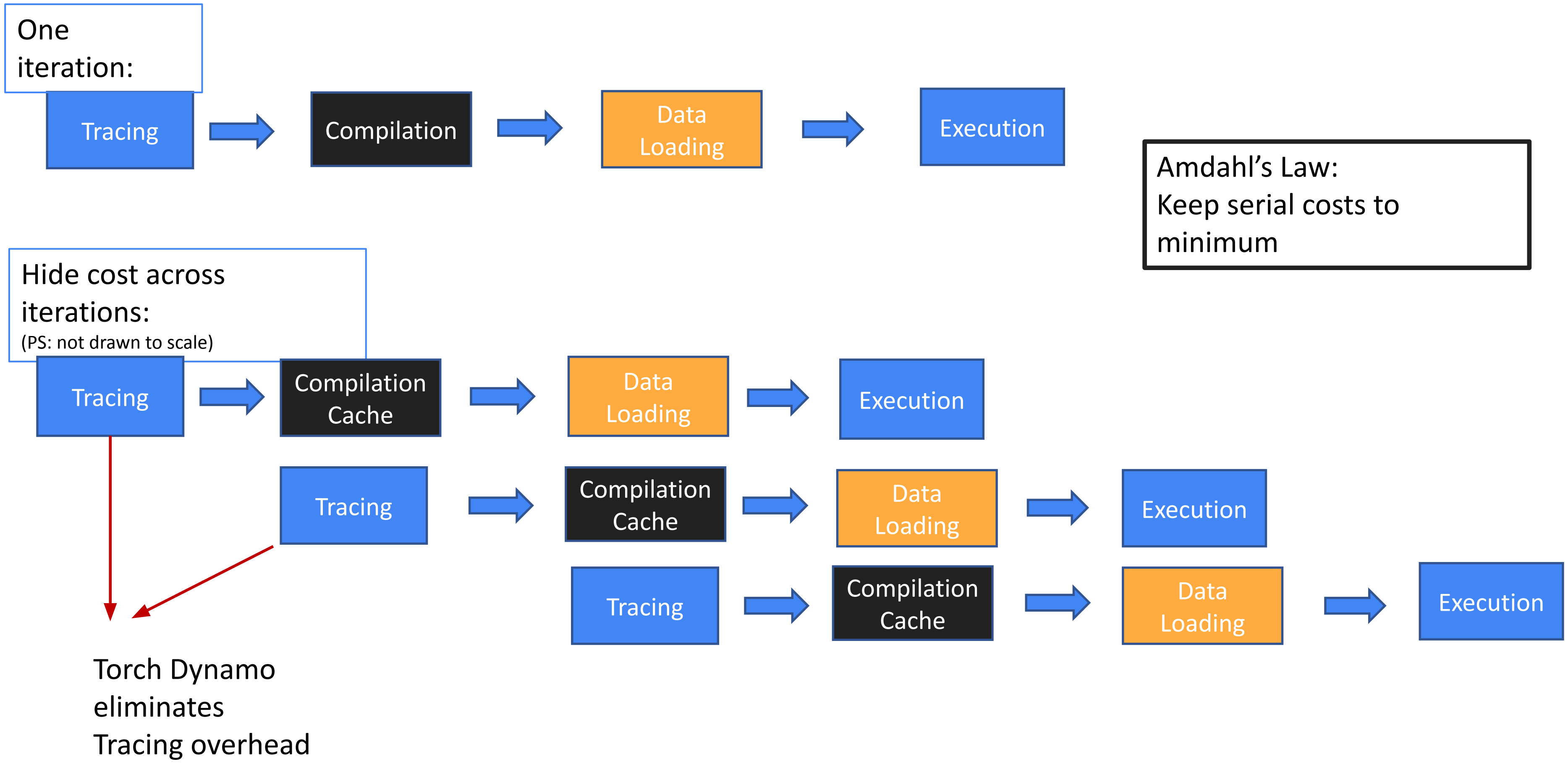
- Optimize PyTorch models across breadth for multiple domains (NLP, Vision..) for Trainium
- Scalability to very large parameter models (> 175Billion parameters) with very good utilization of the 3.36 Pflops/instance of bf16 raw performance
- Seamless transition to run large scale PyTorch models end-to-end with no/very few changes

Lowering Flow



- Typically generate 3 types of HLO graphs: initialization, Fwd/Bwd and Optimizer
- Multiple Neuron Compiler backend optimizations including custom ops
- Compute and Communications ops optimized together (separate CC engine to run in parallel to compute ops)
- SPMD “vs” MPMD

Execution Flow:

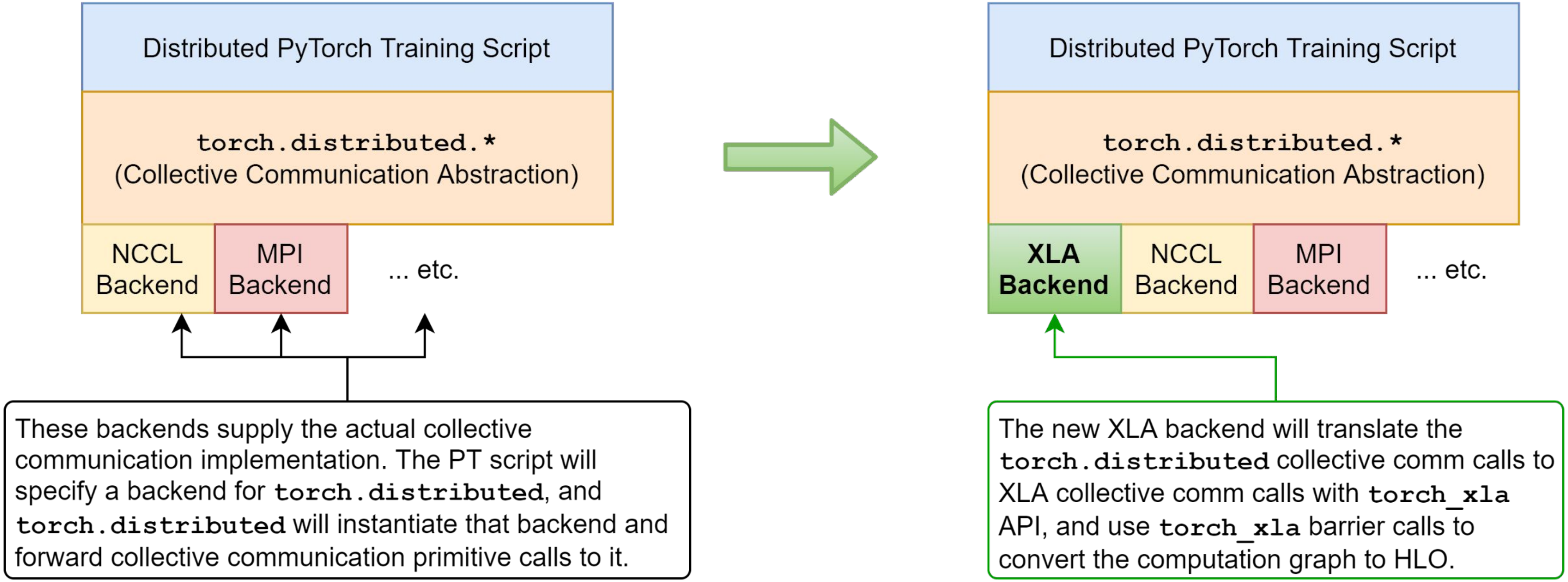


XLA based GPT3, BERT scale out

- GPT (Megatron-LM*) and BERT (HF**) successfully scaled on Trainium1 Clusters
  - https://awsdocs-neuron-staging.readthedocs-hosted.com/en/api_guide/general/benchmarks/trn1/trn1-performance.html#trn1-performance
- Migrating existing code to XLA
  - Single worker:
    - Fairly smooth, HuggingFace models using device = xla_device()
    - Minor incompatibility issues at PyTorch API due to CUDA idioms (Megatron-LM)
    - Convert fused CUDA operators to generic PyTorch API
    - Specialized XLA centric data loaders to overlap tracing, compilation and data batching
  - Multi worker/Distributed parallelism:
    - Gaps lowering from PyTorch torch distributed API to XLA HLO
    - Add new “XLA” backend to the existing PyTorch distributed backends

* a) https://github.com/aws-neuron/aws-neuron-reference-for-megatron-lm
* b) https://awsdocs-neuron.readthedocs-hosted.com/en/latest/frameworks/torch/torch-neuronx/tutorials/training/megatron_lm_gpt.html
** c) https://awsdocs-neuron.readthedocs-hosted.com/en/latest/frameworks/torch/torch-neuronx/tutorials/training/bert.html

XLA Backend for torch distributed operators



XLA Collective Communication

- Trainium Communication (Neuron CCOM) infrastructure supports multiple process groups
  - Helps create 2D/3D mesh topologies, supporting Model Parallelism (Tensor + Pipeline) + Data Parallelism
  - Direct compatibility with PyTorch Group API
    - Caveat: Creates MPMD style graphs, as we plug in only local group list
    - Workaround: ”Infer” a global mesh so that SPMD can be brought back
- Distributed libraries (DDP, FSDP, DeepSpeed) :
  - FSDP: Very Useful as it can wrap a PyTorch Model using a user provided ”wrapping policy”
  - Optimization of FSDP over XLA: Prefetching, allgather, reduce-scatter

Future Plans

- Optimize FSDP at scale
- Migrate to PyTorch 2.0 (https://pytorch.org/get-started/pytorch-2.0/)
  - Torch Dynamo
  - New PjRT
  - Dynamic Shapes
- SPMD (https://github.com/pytorch/xla/issues/3871)
- Support more advanced models (MoE, Diffusion)
- Support more communication patterns (AlltoAll)

Community News

Community Updates:

- Reminder: IREE moving into OpenXLA Project, more details to come
- RFCs:
  - OpenXLA PJRT Plugin (open for comments)
  - Handling XLA exceptions in Python (in review)
- Project governance proposal v1
  - To be merged this week
  - Main points of feedback:
    - Defining governance scope
    - Use of rank-choice voting
    - Opening committer role up over time

Let's continue to discuss on GitHub!

github.com/openxla/xla/discussions

OpenXLA