TensorFlow w/XLA: TensorFlow, Compiled!
Expressiveness with performance

Pre-release Documentation (or search GitHub repository for ‘XLA’):
https://www.tensorflow.org/versions/master/resources/xla_prerelease.html

Jeff Dean
Google Brain team
g.co/brain
presenting work done by the XLA team and Google Brain team
It takes a village to raise a compiler.

- Ancient proverb
Why Did We Build TensorFlow?

Wanted system that was **flexible**, **scalable**, and **production-ready**

DistBelief, our first system, was good on two of these, but lacked **flexibility**

Most existing open-source packages were also good on 2 of 3 but not all 3
TensorFlow Goals

Establish **common platform** for expressing machine learning ideas and systems

Make this platform the **best in the world** for both research and production use

Open source it so that it becomes a **platform for everyone**, not just Google
Facts and Figures

Launched on Nov. 9, 2015

Reasonably fully-featured:
  auto differentiation, queues, control flow, fairly comprehensive set of ops, ...

Tutorials made system accessible

Out-of-the-box support for CPUs, GPUs, multiple devices, multiple platforms
Some Stats

500+ contributors, most of them outside Google

11,000+ commits since Nov, 2015

1M+ binary downloads

#16 most popular repository on GitHub by stars

Used in ML classes at quite a few universities now:
   Toronto, Berkeley, Stanford, …

Many companies/organizations using TensorFlow:
   Google, DeepMind, OpenAI, Twitter, Snapchat, Airbus, Uber, …
TensorFlow Strengths

Flexible
Expressive
Extensible
Just-In-Time Compilation
via XLA, "Accelerated Linear Algebra" compiler

TF graphs go in,

Optimized & specialized assembly comes out.

Let's explain that!
Demo:
Inspect JIT code in TensorFlow iPython shell

```python
In [1]: %cpaste
Pasting code: enter '---' alone on the line to stop or use Ctrl-D.

```x = tf.placeholder(tf.float32, [4])
```with tf.device("device:XLA_CPU:0"):
```  y = x * x
```  result = sess.run(y, {x: [1.5, 0.5, -0.5, -1.5]})
``````
What's **JIT** all about?

- Program built at runtime
- Low-overhead compilation
- Dim variables (e.g. batch size) can bind very late
- Prototype w/freedom of TF development
TF-Level Block Diagram

- TensorFlow
- Existing TensorFlow Core
- TF Auto-JIT
- TF CPU Ops
- TF GPU Ops
- TF TPU Ops
- XLA
- XLA:CPU
- XLA:GPU
- XLA:TPU

Target graphs explicitly at an XLA "device"
TF-Level Block Diagram

TF CPU Ops | TF GPU Ops | TF TPU Ops | XLA
---|---|---|---
Existing TensorFlow Core | TF Auto-JIT | XLA:CPU | XLA:GPU | XLA:TPU

Or let TF find JIT- compilable op clusters for you!
TF-Level Block Diagram

Things that don't compile can still be placed on existing devices

- TensorFlow
- Existing TensorFlow Core
  - TF CPU Ops
  - TF GPU Ops
  - TF TPU Ops
- TF Auto-JIT
- XLA
  - XLA:CPU
  - XLA:GPU
  - XLA:TPU
Complementary Attributes!

- Flexible
- Expressive
- Extensible

Interpreted
- Dynamic
- Stateful

"Black-Box" Modular

Compiled
- Static
- Pure
- Primitives

Think & write this way...

But get optimization benefits of these!
What has us excited?

Server-side speedups

XLA's **JIT compilation** and **specialization**

Significant performance wins

SyntaxNet latency reductions: 200µs ⇒ 5µs (extreme case)
What has us excited?
Mobile footprint reductions

XLA's Ahead-of-Time compilation
Turn models to executables
  Eliminates much of TensorFlow runtime
  Cross-compile for ARM, PPC, x86
LSTM model for mobile: ~1MB ⇒ 10s of KBs
What has us excited?
Whole-Program Analysis made easy

XLA's **High-Level Optimizer**
Reusable toolkit of global optimizations
Layout (e.g. dim order, cache-line padding) is parameterized
Mix & match platform-agnostic & target specific passes
Caveats?
It's still early days!

Best time to start the dialogue :-)

Not all TensorFlow ops compile
Wins accumulating day by day, not everything is faster yet
Haven't devoted equal time to all platforms
With the community we believe we could do much more!
Open source release in O(1 month)

Note: some won't compile by design (e.g. DynamicStitch)
(That being said...)

Benchmark Results

TF:XLA:GPU vs TF:GPU
Increasing complexity from "toy demo" to "large, complex neural nets"...

XLA gives 30% speedup

XLA gives 20% speedup
Ah, more real!
LSTMs have element-wise ops the compiler "fuses"
More on that later...
Very real: Neural Machine Translation! [https://goo.gl/SzbQCS](https://goo.gl/SzbQCS)
Full-model runs also indicate ~20% speedup
New compiler optimizations tend to benefit across many models

Yay!

XLA gives 20% speedup
Compilation benefits

Specializes the code for your computation

- Eliminates op dispatch overhead
- Fuses ops: avoids round trips to memory
- Analyzes buffers: reuses memory, updates in-place
- Unrolls, vectorizes via known dimensions

↓ executable size: generate what you need!
Under the Hood
XLA program = static, decomposed TF ops

Math-looking **primitive ops**
Make **macro-ops by composition**
Supports many neural net definitions
Classic TensorFlow example

biases
weights
examples
labels

MatMul
Add
Relu
Softmax

Math! We get it.
Classic TensorFlow example

biases
weights
examples
labels

MatMul
Add
Max(0.0, _)
Softmax

Mathier!
Classic TensorFlow example

biases
weights
examples
labels

MatMul
Add
Max(0.0, _)

Softmax

Aha, one of these things is not like the others...
A key question:
Why write every new macro-op in C++?
Why can't we just compose them out of existing TF ops?

An answer: you don't want to pay a performance penalty.

But, what if op composition had the performance of C++?
auto weighted = Dot(input, weights);
auto weighted_sum = Add(weighted, biases, /*broadcast=*/{1});
auto max_activation = Reduce(
    weighted_sum, Constant(MinValue(F32)), Max, /*reduce_dims=*/{1});
auto activations_normalized =
    Exp(Sub(weighted_sum, max_activation, /*broadcast=*/{0}));
auto activations_sum =
    Reduce(activations_normalized, Constant(0.0f), Add, /*reduce_dims=*/{1});
auto predicted = Div(activations_normalized,
    activations_sum, /*broadcast=*/{0});
Automatic Operation Fusion
XLA composes & specializes primitive operations

Note: this is all expressible in TensorFlow
Not done due to performance concerns
XLA removes the performance concern
Avoids combinatorial explosion of op fusions
(e.g. for custom LSTM cell)

macro-ops * primitives * dim sizes * backends * devices!
XLA APIs
(never seen by normal TensorFlow users)
XLA Block Diagram

TensorFlow

ComputationBuilder API

High-Level Optimizer (HLO):
Target Independent

Builds "HLO IR"

Lowering to "LLO IR"

Low-Level Optimizer (LLO):
Target Specific

In-Memory Executable Object

Executor API

Code Cache

TransferManager

StreamExecutor

Assembled code generation
XLA is Designed for Reuse
Retargetability & pragmatism

Pluggable backends
HLO pass "toolkit"
Can emit calls to libraries like BLAS or CuDNN

Either use LLVM

Or Bring-Your-Own Low Level Optimizer
Minimal XLA backend:
An LLVM pipeline
A StreamExecutor plugin
Let's instantiate for different platforms!

TensorFlow

ComputationBuilder API

High-Level Optimizer (HLO)

Low-Level Optimizer (LLO)

Executor API

In-Memory Executable Object

Code Cache

TransferManager

StreamExecutor
XLA:CPU

TensorFlow

ComputationBuilder API

High-Level Optimizer (HLO)

LLVM:$TARGET

In-Memory Executable Object

In-memory {ARM, PPC, x86} JIT blob

Executor API

Code Cache

TransferManager

StreamExecutor:Host
XLA:GPU:OpenCL

- **TensorFlow**
- **ComputationBuilder API**
- **High-Level Optimizer (HLO)**
- **LLVM:$TARGET**
- **In-Memory Executable Object**
- **Executor API**
  - **Code Cache**
  - **TransferManager**
  - **StreamExecutor:OpenCL**

**Key Components**:
- In-memory kernels & library calls
- Executor API

**Tools**:
- TensorFlow
- ComputationBuilder API
- High-Level Optimizer (HLO)
- LLVM:$TARGET
{CPU, GPU} HLO pipeline; one slide each
Mixes target-independent passes & dependent passes in a pipeline

```cpp
HloPassPipeline pipeline("CPU");
pipeline.AddPass<Inliner>();
    .AddPass<ConvCanonicalization>()
    .AddPass<HloPassFix<ReshapeMover>>()
    .AddPass<HloSubcomputationUnification>()
    .AddPass<HloCSE>(/*is_layout_sensitive=*/false)
    .AddPass<CpuInstructionFusion>()
    .AddPass<CpuLayoutAssignment>();
    .AddPass<HloPassFix<AlgebraicSimplifier>>(
        /*is_layout_sensitive=*/true, /*add_bitcasts=*/true)
    .AddPass<HloCSE>(/*is_layout_sensitive=*/true)
    .AddPass<CopyInsertion>()
    .AddPass<ParallelizationPreparation>();
pipeline.Run(hlo_module);
```
HloPassPipeline pipeline("GPU");
pipeline.AddPass<ConvolutionFolding>()
  .AddPass<ReshapeMover>().AddPass<TransposeFolding>()
  .AddPass<HloSubcomputationUnification>()
  .AddPass<HloCSE>(/*is_layout_sensitive=*/false)
  .AddPass<HloPassFix<ReduceFactorizer>)(
    device_desc.threads_per_core_limit() * device_desc.core_count())
  .AddPass<HloPassFix<AlgebraicSimplifier>>(false)
  .AddPass<ReduceSplitter>()
  .AddPass<GpuInstructionFusion>(/*may_duplicate=*/false)
  .AddPass<PadInsertion>().AddPass<GpuLayoutAssignment>()
  .AddPass<HloPassFix<AlgebraicSimplifier>>(
    /*is_layout_sensitive=*/true, /*add_bitcasts=*/true)
  .AddPass<HloCSE>(/*is_layout_sensitive=*/true).AddPass<GpuCopyInsertion>();
pipeline.Run(hlo_module);
XLA: Prototype to Deployment
Potential at various phases of the lifecycle

- **JIT compilation** when prototyping
- **Compilation caching** as you scale
- **AoT compilation** for mobile/embedded & latency

Control & observe **static properties** of the program

E.g. peak memory usage
Future Work

ALWAYS MORE PERFORMANCE!
Multi-device-targeting compilation
Cross-layer optimizations
Sparse operation support
Feedback-directed opt & auto-tuning
Conclusions: XLA release for TensorFlow is coming soon!
Performance will improve across the board
Write the code naturally, let compiler deal with performance
Modular infrastructure
Whole-program optimization
Mix compilation & library techniques
Easy to target wide variety of different kinds of HW

Pre-release Documentation (or search TensorFlow GitHub repository for ‘XLA’):
https://www.tensorflow.org/versions/master/resources/xla_prerelease.html
Backup slides in case internet doesn’t work for video
Demo: Inspect **JIT code in TensorFlow**
iPython shell

```
$ shell --xla_dump_assembly=true
TensorFlow shell
In [1]: %cpaste
Pasting code; enter '---' alone on the line to stop or use Ctrl-D.
:with tf.Session() as sess:
 : x = tf.placeholder(tf.float32, [4])
 : with tf.device("device:XLA_CPU:0"):
 : y = x * x
 : result = sess.run(y, {x: [1.5, 0.5, -0.5, -1.5]})

```

```
cluster_3.v4:
0x00000000  movq  (%rdx), %rax
0x00000003  vmovaps (%rax), %xmm0
0x00000007  vmulps %xmm0, %xmm0, %xmm0
0x0000000b  vmovaps %xmm0, (%rdi)
0x0000000f  retq
```

In [2]:

**XLA: CPU**

**XLA: GPU**
Demo: Inspect JIT code in TensorFlow

iPython shell
Demo: Inspect **JIT code** in TensorFlow iPython shell

```
.target sm_35
.address_size 64

.globl _multiply

.visible .entry _multiply(
  .param .u64 _multiply_param_0,
  .param .u64 _multiply_param_1,
  .param .u64 _multiply_param_2,
  .param .u64 _multiply_param_3
)

.maxntid 4, 1, 1
{
  .reg .pred %p<2>;
  .reg .f32 %f<3>;
  .reg .b32 %r<5>;
  .reg .b64 %rd<8>;
  mov.u32 %r2, %ctaid.x;
  mov.u32 %r3, %tid.x;
  shl.b32 %r4, %r2, 2;
  add.s32 %r1, %r4, %r3;
  setp.gt.u32 %p1, %r1, 3;
  @%p1 bra LBB0_2;
}
```