TensorRT和DLA(Deep Learning Accelerator)

NVIDIA DLA（Deep Learning Accelerator--深度学习加速器）是一款针对深度学习操作的固定功能加速器引擎。 DLA 旨在对卷积神经网络进行全硬件加速。 DLA支持卷积、反卷积、全连接、激活、池化、批量归一化等各种层，DLA不支持Explicit Quantization 。

有关 TensorRT 层中 DLA 支持的更多信息，请参阅DLA 支持的层。 `trtexec` 工具具有在 DLA 上运行网络的附加参数，请参阅trtexec 。

`trtexec` 在DLA 上运行 ResNet-50 FP16 网络：

```
./trtexec --onnx=data/resnet50/ResNet50.onnx --useDLACore=0 --fp16 --allowGPUFallback
```

`trtexec` 在DLA 上运行 ResNet-50 INT8 网络:

```
./trtexec --onnx=data/resnet50/ResNet50.onnx --useDLACore=0 --int8 --allowGPUFallback
```

12.1. Running On DLA During TensorRT Inference

TensorRT 构建器可以配置为在 DLA 上启用推理。 DLA 支持目前仅限于在 `FP16` 或 `INT8` 模式下运行的网络。 `DeviceType` 枚举用于指定网络或层在其上执行的设备。 `IBuilderConfig` 类中的以下 API 函数可用于配置网络以使用 DLA：

`setDeviceType(ILayer* layer, DeviceType deviceType)`

此函数可用于设置层必须在其上执行的设备类型

`getDeviceType(const ILayer* layer)`

此函数可用于返回该层执行的设备类型。如果层在 GPU 上执行，则返回`DeviceType::kGPU` 。

`canRunOnDLA(const ILayer* layer)`

此功能可用于检查层是否可以在 DLA 上运行。

`setDefaultDeviceType(DeviceType deviceType)`

此函数设置构建器使用的默认设备类型。它确保可以在 DLA 上运行的所有层都在 DLA 上运行，除非 `setDeviceType` 用于覆盖层的 `deviceType` 。

`getDefaultDeviceType()`

此函数返回由 `setDefaultDeviceType` 设置的默认设备类型.

`isDeviceTypeSet(const ILayer* layer)`

此函数检查是否已为该层显式设置了 `deviceType` 。

`resetDeviceType(ILayer* layer)`

此函数重置此层的 `deviceType` 。如果未指定，该值将重置为由 `setDefaultDeviceType` 或 `DeviceType::kGPU` 指定的设备类型。

`allowGPUFallback(bool setFallBackMode)`

如果应该在 DLA 上运行的层无法在 DLA 上运行，此函数会通知构建器使用 GPU。有关详细信息，请参阅GPU 回退模式。

`reset()`

此函数可用于重置`IBuilderConfig`状态，它将所有层的`deviceType`设置为`DeviceType::kGPU`。重置后，构建器可以重新用于构建具有不同 DLA 配置的另一个网络。

`IBuilder`类中的以下 API 函数可用于帮助配置网络以使用 DLA：

`getMaxDLABatchSize()`

此函数返回 DLA 可以支持的最大批量大小。

注意：对于任何张量，索引维度的总体积加上请求的批量大小不得超过此函数返回的值。

`getNbDLACores()`

此函数返回用户可用的 DLA 内核数。

如果构建器不可访问，例如在推理应用程序中在线加载计划文件的情况下，则可以通过对`IRuntime`使用 DLA 扩展以不同方式指定要使用的DLA。`IRuntime`类中的以下 API 函数可用于配置网络以使用 DLA：

`getNbDLACores()`

此函数返回用户可访问的 DLA 内核数。

`setDLACore(int dlaCore)`

要在其上执行的 DLA 内核。其中`dlaCore`是介于0和`getNbDLACores() - 1`之间的值。默认值为0

`getDLACore()`

运行时执行分配到的 DLA 核心。默认值为 0。

12.1.1. Example: sampleMNIST With DLA

本节提供有关如何在启用 DLA 的情况下运行 TensorRT 示例的详细信息。

位于 GitHub 存储库中的`sampleMNIST`演示了如何导入经过训练的模型、构建 TensorRT 引擎、序列化和反序列化引擎，最后使用引擎执行推理。

该示例首先创建构建器：

```
auto builder = SampleUniquePtr<nvinfer1::IBuilder>
(nvinfer1::createInferBuilder(gLogger));
if (!builder) return false;
builder->setMaxBatchSize(batchSize);
```

然后，启用GPUFallback模式：

```
config->setFlag(BuilderFlag::kGPU_FALLBACK);
config->setFlag(BuilderFlag::kFP16); or config->setFlag(BuilderFlag::kINT8);
```

在 DLA 上启用执行，其中dlaCore指定要在其上执行的 DLA 内核：

```
config->setDefaultDeviceType(DeviceType::kDLA);
config->setDLACore(dlaCore);
```

通过这些额外的更改，sampleMNIST 已准备好在 DLA 上执行。要使用 DLA Core 1 运行 sampleMNIST，请使用以下命令：

```
 ./sample_mnist --useDLACore=0 [--int8|--fp16]
```

12.1.2. Example: Enable DLA Mode For A Layer During Network Creation

在这个例子中，让我们创建一个包含输入、卷积和输出的简单网络。

1.创建构建器、构建器配置和网络：

```
IBuilder* builder = createInferBuilder(gLogger);
IBuilderConfig* config = builder.createBuilderConfig();
INetworkDefinition* network = builder->createNetworkV2(0U);
```

2.使用输入维度将输入层添加到网络。

```
auto data = network->addInput(INPUT_BLOB_NAME, dt, Dims3{1, INPUT_H, INPUT_W});
```

3.添加具有隐藏层输入节点、步幅和权重的卷积层以用于卷积核和偏差。

```
auto conv1 = network->addConvolution(*data->getOutput(0), 20, DimsHW{5, 5},
weightMap["conv1filter"], weightMap["conv1bias"]);
conv1->setStride(DimsHW{1, 1});
```

4.将卷积层设置为在 DLA 上运行：

```
if(canRunOnDLA(conv1))
{
config->setFlag(BuilderFlag::kFP16); or config->setFlag(BuilderFlag::kINT8);
builder->setDeviceType(conv1, DeviceType::kDLA);

}
```

5.标记输出

```
network->markOutput(*conv1->getOutput(0));
```

6.将 DLA 内核设置为在以下位置执行：

```
config->setDLACore(0)
```

12.2. DLA Supported Layers

本节列出了 DLA 支持的层以及与每个层相关的约束。

在 DLA 上运行时的一般限制（适用于所有层）

- 支持的最大批量大小为 4096。
- DLA 不支持动态尺寸。因此，对于通配符维度，配置文件的`min`、`max`和`opt`值必须相等。
- 如果违反了任何限制，并且启用了`GpuFallback`，TensorRT可以将DLA网络分成多个部分。否则，TensorRT会发出错误并返回。更多信息，请参考GPU回退模式。
- 由于硬件和软件内存限制，最多可以同时使用四个 DLA 可加载项。

注意： DLA 的批量大小是除CHW维度之外的所有索引维度的乘积。例如，如果输入维度为NPQRS，则有效批量大小为N*P 。

层特定限制

卷积层和全连接层

- 仅支持两个空间维度操作。
- 支持 FP16 和 INT8。
- 内核大小的每个维度都必须在[1, 32]范围内。
- 填充(Padding)必须在[0, 31]范围内。
- 填充的维度必须小于相应的内核维度。
- 步幅的尺寸必须在[1, 8]范围内。
- 输出映射的数量必须在[1, 8192]范围内。
- 对于使用格式`TensorFormat::kLINEAR`、`TensorFormat::kCHW16`和`TensorFormat::kCHW32`的操作，组数必须在[1, 8192]范围内。
- 对于使用格式`TensorFormat::kCHW4`的操作，组数必须在[1, 4]范围内。
- 空洞卷积(Dilated convolution)必须在[1, 32]范围内。
- 如果 CBUF 大小要求`wtBanksForOneKernel + minDataBanks`超过`numConvBufBankAllotted`限制`16`，则不支持操作，其中`CBUF`是在对输入权重和激活进行操作之前存储输入权重和激活的内部卷积缓存，`wtBanksForOneKernel`是一个内核存储最小权重/卷积所需的核元素，`minDataBanks`是存储卷积所需的最小激活数据的最小库。伪代码细节如下：

```
wtBanksForOneKernel = uint32(ceil(roundUp(inputDims_c * kernelSize_h *
kernelSize_w * (INT8 ? 1 : 2), 128) / 32768.0))

minDataBanks = uint32(ceil(float(entriesPerDataSlice * dilatedKernelHt) /
256.0)) where entriesPerDataSlice = uint32(ceil(ceil(inputDims_c * (INT8 ? 1 :
2) / 32.0) * inputDims_w / 4.0)) and dilatedKernelHt = (kernelSize_h - 1) *
dilation_h + 1

FAIL if wtBanksForOneKernel + minDataBanks > 16, PASS otherwise.
```

反卷积层

- 仅支持两个空间维度操作。
- 支持`FP16`和`INT8`。
- 除了`1x[64, 96, 128]`和`[64, 96, 128]x1`之外，内核的尺寸必须在`[1, 32]`范围内。
- TensorRT 在 DLA 上禁用了反卷积平方内核并在`[23 - 32]`范围内跨步，因为它们显着减慢了编译速度。
- 填充(Padding)必须为0 。
- 分组反卷积必须为1 。
- 扩张反卷积必须为1 。
- 输入通道数必须在[1, 8192]范围内。
- 输出通道数必须在[1, 8192]范围内。

池化层

- 仅支持两个空间维度操作。
- 支持 `FP16` 和 `INT8`。
- 支持的操作： `kMAX` ， `kAVERAGE` 。
- 窗口的尺寸必须在 `[1， 8]` 范围内。
- 填充的尺寸必须在 `[0， 7]` 范围内。
- 步幅的尺寸必须在[1, 16]范围内。
- 使用 `INT8` 模式，输入和输出张量标度必须相同。

激活层

- 仅支持两个空间维度操作。
- 支持 `FP16` 和 `INT8`。
- 支持的函数： `ReLU` 、 `Sigmoid` 、 `TanH` 、 `Clipped ReLU` 和 `Leaky ReLU` 。
  - ReLU不支持负斜率。
  - Clipped ReLU仅支持[1, 127]范围内的值。
  - TanH , Sigmoid INT8 支持通过自动升级到 FP16 来支持。

参数 ReLU 层

- 斜率输入必须是构建时间常数。

ElementWise 层

- 仅支持两个空间维度操作。
- 支持 `FP16` 和 `INT8`。
- 支持的操作： `Sum` 、 `Sub` 、 `Product` 、 `Max` 和 `Min` 。
-

注意：在 Xavier 上，TensorRT 将 DLA Scale 层和 DLA ElementWise 层与操作 Sum连接以支持Sub操作，单个 Xavier DLA ElementWise 层不支持。

Scale层

- 仅支持两个空间维度操作。
- 支持 `FP16` 和 `INT8`。
- 支持的模式： `Uniform` 、 `Per-Channel` 和 `ElementWise` 。
- 仅支持缩放和移位操作。

LRN（局部响应归一化）层

- 允许的窗口大小为3 、 5 、 7或9 。
- 支持的规范化区域是ACROSS_CHANNELS 。
- LRN INT8。

连接层

- DLA 仅支持沿通道轴连接。
- Concat 必须至少有两个输入。
- 所有输入必须具有相同的空间维度。
- 对于 INT8 模式，所有输入的动态范围必须相同。
- 对于 INT8 模式，输出的动态范围必须等于每个输入。

Resize层

- 刻度的数量必须正好是4 。
- scale 中的前两个元素必须正好为1 （对于未更改的批次和通道尺寸）。
- scale 中的最后两个元素，分别表示沿高度和宽度维度的比例值，在最近邻模式下需要为[1, 32]范围内的整数值，在双线性模式下需要为[1, 4]范围内的整数值。

Unary 层

- 仅支持 ABS 操作。

Softmax 层

- 仅支持 NVIDIA Orin™，不支持 Xavier™。
- 仅支持批量大小为 1 的单个输入。
- 输入的非批量、非轴维度都应该是大小 1。例如，对于轴 = 1 的 softmax（即在 C 维度上），H 和 W 维度的大小都应该是 1。

注意：当使用 TensorRT 在 DLA 上运行 INT8 网络时，建议将操作添加到同一子图中，以通过允许它们融合并为中间结果保留更高的精度来减少在 DLA 上运行的网络的子图上的量化误差。通过将张量设置为网络输出张量来拆分子图以检查中间结果可能会由于禁用这些优化而导致不同级别的量化误差。

12.3. GPU Fallback Mode

如果被标记为在DLA上运行的层不能在DLA上运行，则 `GPUFallbackMode` 设置生成器使用GPU。

由于以下原因，层无法在 DLA 上运行：

1. DLA 不支持层操作。
2. 指定的参数超出了 DLA 支持的范围。
3. 给定的批量大小超过了允许的最大 DLA 批量大小。有关详细信息，请参阅DLA 支持的层。
4. 网络中的层组合导致内部状态超过 DLA 能够支持的状态。
5. 平台上没有可用的 DLA 引擎。

如果 `GPUFallbackMode` 设置为 `false` ，则设置为在 DLA 上执行但无法在 DLA 上运行的层会导致错误。但是，将 `GPUFallbackMode` 设置为 `true` 后，它会在报告警告后继续在 GPU 上执行。

同样，如果 `defaultDeviceType` 设置为 `DeviceType::kDLA` 并且 `GPUFallbackMode` 设置为 `false` ，则如果任何层无法在 DLA 上运行，则会导致错误。将 `GPUFallbackMode` 设置为 `true` 时，它会报告警告并继续在 GPU 上执行。

如果网络中的层组合无法在 DLA 上运行，则组合中的所有层都在 GPU 上执行。

12.4. I/O Formats on DLA

DLA 支持设备独有的格式，并且由于矢量宽度字节要求而对其布局有限制。

对于 DLA 输入，支持 `kDLA_LINEAR ( FP16 , INT8 )` 、 `kDLA_HWC4 ( FP16 , INT8 )` 、 `kCHW16 ( FP16 )` 和 `kCHW32 ( INT8 )` 。对于 DLA 输出，仅支持 `kDLA_LINEAR ( FP16 , INT8 )` 、 `kCHW16 ( FP16 )` 和 `kCHW32 ( INT8 )` 。对于 `kCHW16` 和 `kCHW32` 格式，如果C不是整数倍，则必须将其填充到下一个 32 字节边界。

对于 `kDLA_LINEAR` 格式，沿W维度的步幅必须最多填充 64 个字节。内存格式等效于维度为 `[N][C][H][roundUp(W, 64/elementSize)]` 的C数组，其中 `FP16` 的 `elementSize` 为 2， `Int8` 为 1 ，张量坐标为 `(n, c, h, w)` 映射到数组下标 `[n][c][h][w]` 。

对于`kDLA_HWC4`格式，沿`W`维度的步幅必须是 Xavier 上 32 字节和 Orin 上 64 字节的倍数。

- 当`C == 1`时，TensorRT 将格式映射到本机灰度图像格式。
- 当`C == 3`或`C == 4`时，它映射到本机彩色图像格式。如果`C == 3`，沿`W`轴步进的步幅需要填充为 4 个元素。

  在这种情况下，填充通道位于第 4 个索引处。理想情况下，填充值无关紧要，因为权重中的第 4 个通道被 DLA 编译器填充为零；但是，应用程序分配四个通道的零填充缓冲区并填充三个有效通道是安全的。
- 当`C`为`{1, 3, 4}`时，填充后的`C'`分别为`{1, 4, 4}`，内存布局等价于维度为`[N][H][roundUp(W, 32/C'/elementSize)][C']`的C数组, 其中`elementSize`对于`FP16`为 2，对于`Int8`为 1 。张量坐标`(n, c, h, w)`映射到数组下标`[n][h][w][c]`，`roundUp`计算大于或等于`W`的`64/elementSize`的最小倍数。

使用`kDLA_HWC4`作为 DLA 输入格式时，有以下要求：

- C必须是1 、 3或4
- 第一层必须是卷积。
- 卷积参数必须满足 DLA 要求，请参阅DLA Supported Layers 。

当`EngineCapability`为`EngineCapability::kSTANDARD`且 TensorRT 无法为给定的输入/输出格式生成无重构网络时，可以自动将不支持的 DLA 格式转换为支持的 DLA 格式。例如，如果连接到网络输入或输出的层不能在 DLA 上运行，或者如果网络不满足其他 DLA 要求，则插入重新格式化操作以满足约束。在所有情况下，TensorRT 期望数据格式化的步幅都可以通过查询`IExecutionContext::getStrides`来获得。

12.5. DLA Standalone Mode

如果您使用单独的 DLA 运行时组件，则可以使用`EngineCapability::kDLA_STANDALONE`生成 DLA 可加载项。请参阅相关 DLA 运行时组件的文档以了解如何使用可加载项。

当使用`kDLA_STANDALONE`时，TensorRT 为给定的输入/输出格式生成一个无重新格式化的网络。对于 DLA 输入，支持`kLINEAR ( FP16 , INT8 )`、`kCHW4 ( FP16 , INT8 )`、`kCHW16 ( FP16 )`和`kCHW32 ( INT8 )`。而对于 DLA 输出，仅支持`kLINEAR ( FP16 , INT8 )`、`kCHW16 ( FP16 )`和`kCHW32 ( INT8 )`。对于`kCHW16`和`kCHW32`格式，建议C通道数等于向量大小的正整数倍。如果C不是整数倍，则必须将其填充到下一个 32 字节边界。

12.6. Customizing DLA Memory Pools

您可以自定义分配给网络中每个可加载的 DLA 的内存池的大小。共有三种类型的 DLA 内存池（有关每个池的描述，请参见枚举类`MemoryPoolType`）：

- Managed SRAM
- Local DRAM
- Global DRAM

对于每种池类型，使用 API `IBuilderConfig::setMemoryPoolLimit`和`IBuilderConfig::getMemoryPoolLimit`来设置和查询相关池的大小，以便为每个可加载的 DLA 分配更大的内存池。每个可加载的实际需要的内存量可能小于池大小，在这种情况下将分配较小的量。池大小仅用作上限。

请注意，所有 DLA 内存池都需要大小为 2 的幂，最小为 4 KiB。违反此要求会导致 DLA 可加载编译失败。

Managed SRAM 与其他 DRAM 池的区别主要在于角色的不同。以下是Managed SRAM 的一些值得注意的方面：

- 它类似于缓存，因为资源稀缺，DLA 可以通过回退到本地 DRAM 来运行而无需它。
- 任何分配往往都会被充分利用。因此，报告的 SRAM 通常与分配的 SRAM 池的数量相同（在某些情况下可能与用户指定的大小不同）。
- 由于类似于缓存的性质，DLA 在 SRAM 不足时会回退到 DRAM，而不是失败。因此，如果可以负担得起，即使在成功的引擎构建之后也尝试增加 SRAM 的数量，以查看推理速度是否有任何提升。这尤其适用于卸载许多子图的网络。
- Orin 和 Xavier 在每个内核可用的最大 SRAM 数量方面存在差异：Xavier 在 4 个内核（包括 2 个 DLA 内核）中提供总共 4 MiB 的 SRAM，而 Orin 为每个 DLA 内核专用 1 MiB SRAM。这意味着当在一个设备上运行多个网络时，Xavier 需要明确控制总体 SRAM 消耗，而 Orin 在这方面不必担心。

在多子图情况下，重要的是要记住池大小适用于每个 DLA 子图，而不是整个网络。