TensorRT的命令行程序

A.3.1. trtexec

示例目录中包含一个名为`trtexec`的命令行包装工具。 `trtexec`是一种无需开发自己的应用程序即可快速使用 TensorRT 的工具。

`trtexec`工具有三个主要用途：

- 它对于在随机或用户提供的输入数据上对网络进行基准测试很有用。
- 它对于从模型生成序列化引擎很有用。
- 它对于从构建器生成序列化时序缓存很有用。

A.3.1.1. Benchmarking Network

如果您将模型保存为`ONNX`文件、`UFF`文件，或者如果您有`Caffe prototxt`格式的网络描述，则可以使用`trtexec`工具测试使用 TensorRT 在网络上运行推理的性能。 `trtexec`工具有许多选项用于指定输入和输出、性能计时的迭代、允许的精度和其他选项。

为了最大限度地提高 GPU 利用率， `trtexec`会提前将一个batch放入队列。换句话说，它执行以下操作：

```
enqueue batch 0
-> enqueue batch 1
-> wait until batch 0 is done
-> enqueue batch 2
-> wait until batch 1 is done
-> enqueue batch 3
-> wait until batch 2 is done
-> enqueue batch 4
-> ...
```

如果使用多流（`--streams=N`标志），则`trtexec`在每个流上分别遵循此模式。

`trtexec`工具打印以下性能指标。下图显示了`trtexec`运行的示例`Nsight`系统配置文件，其中标记显示了每个性能指标的含义。

Throughput

观察到的吞吐量是通过将执行数除以`Total Host Walltime`来计算的。如果这显着低于 GPU 计算时间的倒数，则 GPU 可能由于主机端开销或数据传输而未被充分利用。使用 CUDA 图（使用`--useCudaGraph`）或禁用 H2D/D2H 传输（使用`--noDataTransfer`）可以提高 GPU 利用率。当`trtexec`检测到 GPU 未充分利用时，输出日志提供了有关使用哪个标志的指导。

Host Latency

`H2D`延迟、GPU 计算时间和`D2H`延迟的总和。这是推断单个执行的延迟。

Enqueue Time

将执行排入队列的主机延迟，包括调用`H2D/D2H` CUDA API、运行主机端方法和启动 CUDA 内核。如果这比 GPU 计算时间长，则 GPU 可能未被充分利用，并且吞吐量可能由主机端开销支配。使用 CUDA 图（带有`--useCudaGraph`）可以减少排队时间。

H2D Latency

单个执行的输入张量的主机到设备数据传输的延迟。添加`--noDataTransfer`以禁用`H2D/D2H`数据传输。

D2H Latency

单个执行的输出张量的设备到主机数据传输的延迟。添加`--noDataTransfer`以禁用`H2D/D2H`数据传输。

GPU Compute Time

为执行 CUDA 内核的 GPU 延迟。

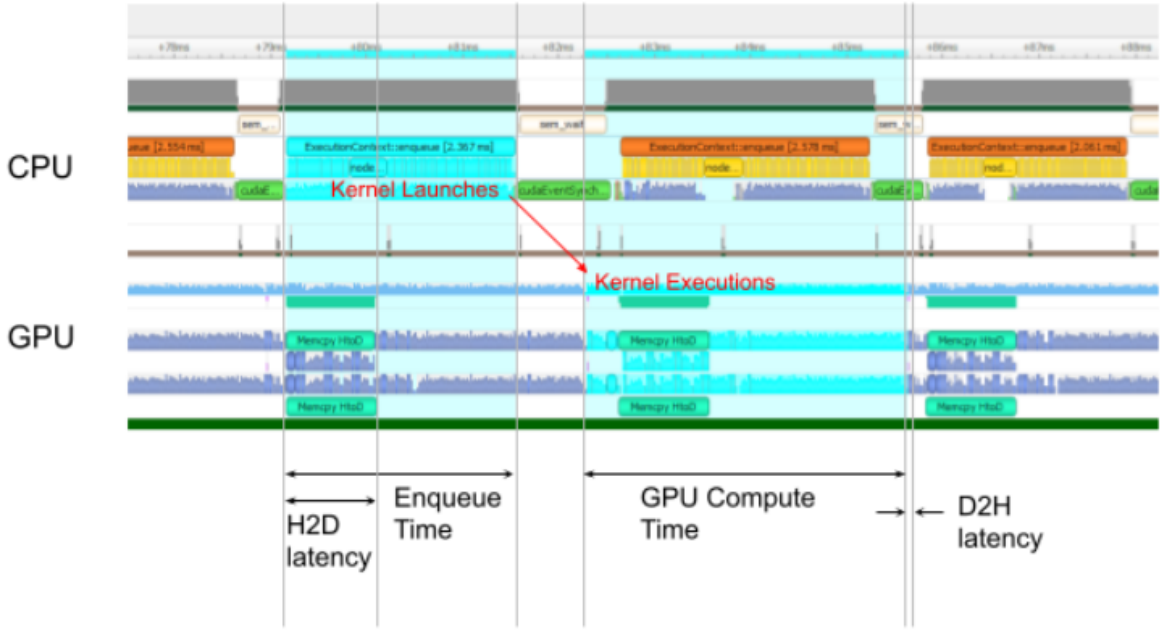
Total Host Walltime

从第一个执行（预热后）入队到最后一个执行完成的主机时间。

Total GPU Compute Time

所有执行的 GPU 计算时间的总和。如果这明显短于`Total Host Walltime`，则 GPU 可能由于主机端开销或数据传输而未得到充分利用。

图 1. 在`Nsight`系统下运行的正常`trtexec`的性能指标（`ShuffleNet`，`BS=16`，best，TitanRTX@1200MHz）



Host latency = (H2D latency) + (GPU Compute latency) + (D2H latency)

Throughput = (Total host wall time for N queries) / N

将`--dumpProfile`标志添加到`trtexec`以显示每层性能配置文件，这使用户可以了解网络中的哪些层在 GPU 执行中花费的时间最多。每层性能分析也适用于作为 CUDA 图启动推理（需要`CUDA 11.1`及更高版本）。此外，使用`--profilingVerbosity=detailed`标志构建引擎并添加`--dumpLayerInfo`标志以显示详细的引擎信息，包括每层详细信息和绑定信息。这可以让你了解引擎中每一层对应的操作及其参数。

A.3.1.2. Serialized Engine Generation

如果您生成保存的序列化引擎文件，您可以将其拉入另一个运行推理的应用程序中。例如，您可以使用 TensorRT 实验室以完全流水线异步方式运行具有来自多个线程的多个执行上下文的引擎，以测试并行推理性能。有一些警告；例如，如果您使用 `Caffe prototxt` 文件并且未提供模型，则会生成随机权重。此外，在 `INT8` 模式下，使用随机权重，这意味着 `trtexec` 不提供校准功能。

A.3.1.3. trtexec

如果您向 `--timingCacheFile` 选项提供时序缓存文件，则构建器可以从中加载现有的分析数据并在层分析期间添加新的分析数据条目。计时缓存文件可以在其他构建器实例中重用，以提高构建器执行时间。建议仅在相同的硬件/软件配置（例如，`CUDA/cuDNN/TensorRT` 版本、设备型号和时钟频率）中重复使用此缓存；否则，可能会出现功能或性能问题。

A.3.1.4. 常用的命令行标志

该部分列出了常用的trtexec命令行标志。

构建阶段的标志

- `--onnx=<model>`： 指定输入 ONNX 模型。
- `--deploy=<caffe_prototxt>`： 指定输入的 Caffe `prototxt` 模型。
- `--uff=<model>`：指定输入 UFF 模型。
- `--output=<tensor>`： 指定输出张量名称。仅当输入模型为 UFF 或 Caffe 格式时才需要。
- `--maxBatch=<BS>`：指定构建引擎的最大批量大小。仅当输入模型为 UFF 或 Caffe 格式时才需要。如果输入模型是 ONNX 格式，请使用 `--minShapes` 、 `--optShapes` 、 `--maxShapes` 标志来控制输入形状的范围，包括批量大小。
- `--minShapes=<shapes>` , `--optShapes=<shapes>` , `--maxShapes=<shapes>`： 指定用于构建引擎的输入形状的范围。仅当输入模型为 ONNX 格式时才需要。
- `--workspace=<size in MB>`： 指定策略允许使用的最大工作空间大小。该标志已被弃用。您可以改用 `--memPoolSize=<pool_spec>` 标志。
- `--memPoolSize=<pool_spec>`： 指定策略允许使用的工作空间的最大大小，以及 DLA 将分配的每个可加载的内存池的大小。
- `--saveEngine=<file>`： 指定保存引擎的路径。
- `--fp16` 、 `--int8` 、 `--noTF32` 、 `--best` ： 指定网络级精度。
- `--sparsity=[disable|enable|force]`： 指定是否使用支持结构化稀疏的策略。
  - `disable`：使用结构化稀疏禁用所有策略。这是默认设置。
  - `enable`： 使用结构化稀疏启用策略。只有当 ONNX 文件中的权重满足结构化稀疏性的要求时，才会使用策略。
  - `force`： 使用结构化稀疏启用策略，并允许 `trtexec` 覆盖 `ONNX` 文件中的权重，以强制它们具有结构化稀疏模式。请注意，不会保留准确性，因此这只是为了获得推理性能。
- `--timingCacheFile=<file>`： 指定要从中加载和保存的时序缓存。
- `--verbose`： 打开详细日志记录。
- `--buildOnly`： 在不运行推理的情况下构建并保存引擎。
- `--profilingVerbosity=[layer_names_only|detailed|none]`： 指定用于构建引擎的分析详细程度。

- `--dumpLayerInfo , --exportLayerInfo=<file>`： 打印/保存引擎的层信息。
- `--precisionConstraints=spec`： 控制精度约束设置。
  - `none`： 没有限制。
  - `prefer`： 如果可能，满足`--layerPrecisions / --layerOutputTypes`设置的精度约束。
  - `obey`：满足由`--layerPrecisions / --layerOutputTypes`设置的精度约束，否则失败。
- `--layerPrecisions=spec`： 控制每层精度约束。仅当PrecisionConstraints设置为服从或首选时才有效。规范是从左到右阅读的，后面的会覆盖前面的。 “ * ”可以用作layerName来指定所有未指定层的默认精度。
  - 例如： `--layerPrecisions=*:fp16,layer_1:fp32`将所有层的精度设置为`FP16`， 除了`layer_1`将设置为`FP32`。
- `--layerOutputTypes=spec`： 控制每层输出类型约束。仅当`PrecisionConstraints`设置为服从或首选时才有效。规范是从左到右阅读的，后面的会覆盖前面的。 “ * ”可以用作`layerName`来指定所有未指定层的默认精度。如果一个层有多个输出，则可以为该层提供用“ + ”分隔的多种类型。
  - 例如： `--layerOutputTypes=*:fp16,layer_1:fp32+fp16`将所有层输出的精度设置为`FP16`， 但`layer_1`除外，其第一个输出将设置为`FP32`，其第二个输出将设置为`FP16`。

推理阶段的标志

- `--loadEngine=<file>`： 从序列化计划文件加载引擎，而不是从输入 ONNX、UFF 或 Caffe 模型构建引擎。
- `--batch=<N>`： 指定运行推理的批次大小。仅当输入模型为 UFF 或 Caffe 格式时才需要。如果输入模型是 ONNX 格式，或者引擎是使用显式批量维度构建的，请改用`--shapes`。
- `--shapes=<shapes>`： 指定要运行推理的输入形状。
- `--warmUp=<duration in ms>`, `--duration=<duration in seconds>`, `--iterations=<N>`：指定预热运行的最短持续时间、推理运行的最短持续时间和推理运行的迭代。例如，设置`--warmUp=0 --duration=0 --iterations`允许用户准确控制运行推理的迭代次数。
- `--useCudaGraph`： 将推理捕获到 CUDA 图并通过启动图来运行推理。当构建的 TensorRT 引擎包含 CUDA 图捕获模式下不允许的操作时，可以忽略此参数。
- `--noDataTransfers`： 关闭主机到设备和设备到主机的数据传输。
- `--streams=<N>`： 并行运行多个流的推理。
- `--verbose`： 打开详细日志记录。
- `--dumpProfile, --exportProfile=<file>`： 打印/保存每层性能配置文件。

有关所有受支持的标志和详细说明，请参阅`trtexec --help`。

有关如何构建此工具及其使用示例的详细信息，请参阅GitHub：trtexec/README.md文件。