深蓝学院
shenlanxueyuan.com

模型推理经验

课程目标

- TRT转换模型的几种方式比较
- TRT如何测试并调优
- CUDA+TensorRT调试建议
- AIer 如何快速上手CUDA
- 模型可以加速到什么程度？以Transformer模型为例

TRT转换模型的几种方式比较

转换方式	代表
转换工具	训练框架自带 TRT 工具：Torch-TensorRT
	第三方转换工具：ONNX-Parser
	TVM (Tensor Virtual Machine)
API方式	TensorRT/demo/BERT
超大Plugin	FastTransformer

方法	易用性	开发成本	性能	灵活性
转换工具	4	2	2	2
API	1	5	4.5	5
超大Plugin	1	3.5	5	4

不同的转换方式适用于不同的场景，没有最好的，只有最适合的。

https://zhuanlan.zhihu.com/p/577735992

TRT转换模型的几种方式比较

举例：

百度搜索首届技术创新挑战赛2022：基于TensorRT的Ernie模型推理加速实践

目标：Ernie模型的极致加速

构建方式	API+plugin构建	超大plugin构建
开源代表	github TensorRT/demo/BERT	FastTransformer
优点	更灵活，适配后续模型修改工作量小	可以尝试更新的优化方式，比如稀疏矩阵乘，INT4，cuda graph
	可以更好的利用TRT的优点：INT8，根据硬件筛选最快实现方式	
缺点	使用优化策略方面可能会受限，比如稀疏矩阵乘，INT4，cuda graph等	不够灵活，二次开发成本高

最适合工业界追求更好性能的构建方式

TRT如何测试并调优

粒度	需求	工具	资料
模型层面	min,opt,max shape 的速度	trtexec	https://www.bilibili.com/video/BV19T4y1e7XK 第2分钟开始
	某个输入维度的速度		
Kernel层面	Kernel的时间	NVIDIA Visual Profiler or Nsight Systems/Compute	https://www.bilibili.com/video/BV13w411o7cu https://www.bilibili.com/video/BV15P4y1R7VG
	Kernel的资源占用		
	Kernel的性能瓶颈		
系统层面	系统函数，比如多线程	Nsight Systems (Nsys)	

TRT如何测试并调优

trtexec输出示例

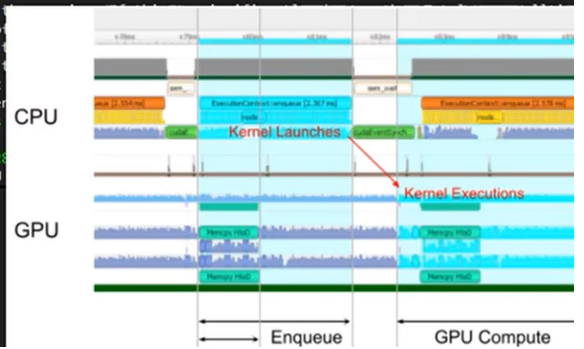
性能测试结果

吞吐量：每秒完成 query 数

各种延迟的统计结果

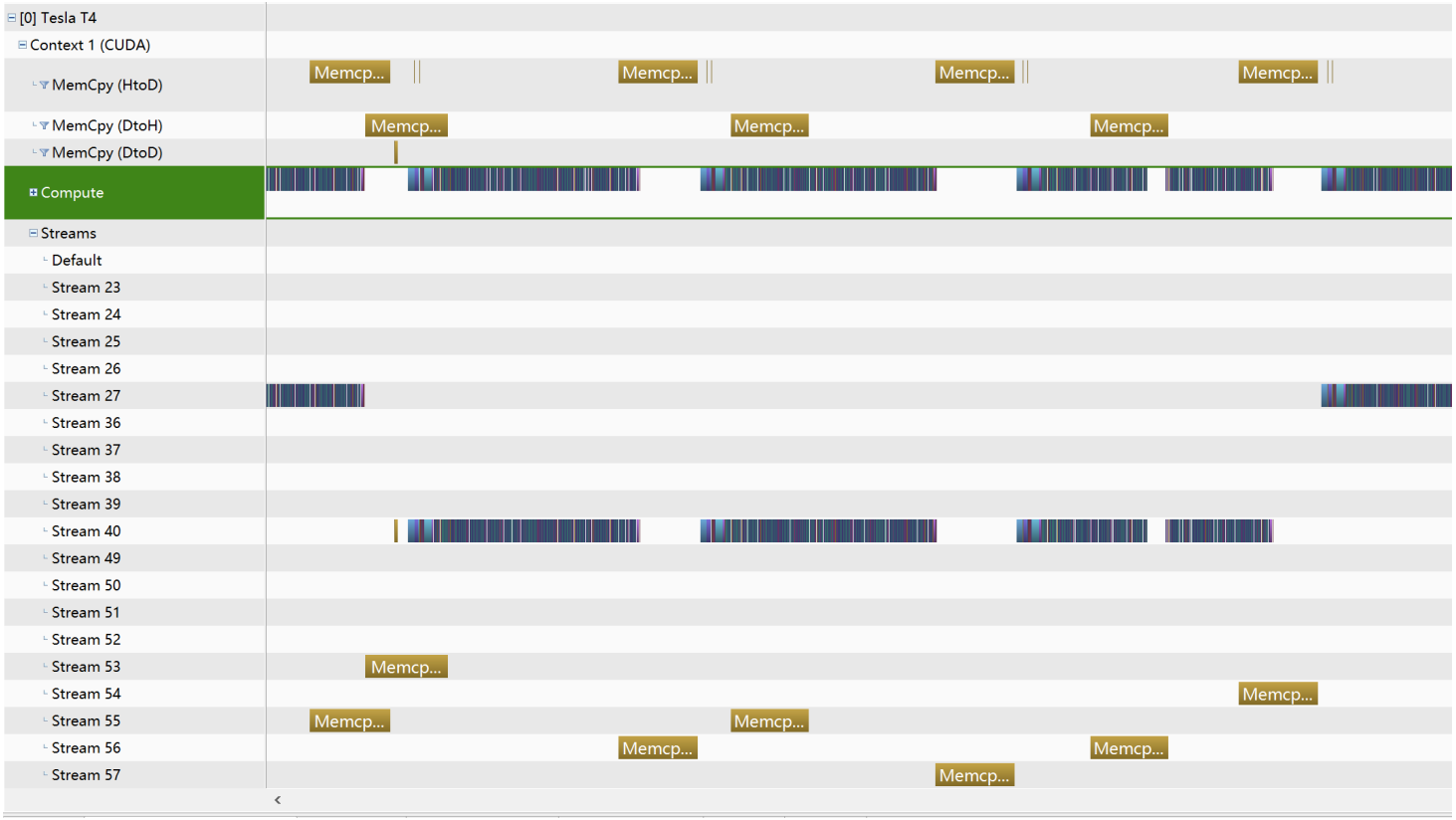
各种延迟的解释

```
[03/07/2022-05:05:52] [I] === Performance summary ===
[03/07/2022-05:05:52] [I] Throughput: 6430.89 qps
[03/07/2022-05:05:52] [I] Latency: min = 0.156982 ms, max = 0.                    s, median = 0.159668 ms, percentile(99%) = 0.165771 ms
[03/07/2022-05:05:52] [I] End-to-End Host Latency: min = 0.171                    ean = 0.284758 ms, median = 0.289673 ms, percentile(99%) = 0.298828 ms
[03/07/2022-05:05:52] [I] Enqueue Time: min = 0.0300293 ms, max = 0.126465 ms, mean = 0.0396717 ms, median = 0.0333252 ms, percentile(99%) = 0.0698242 ms
[03/07/2022-05:05:52] [I] H2D Latency: min = 0.00317383 ms, max = 0.0455933 ms, mean = 0.00408154 ms, median = 0.00366211 ms, percentile(99%) = 0.00854492 ms
[03/07/2022-05:05:52] [I] GPU Compute Time: min = 0.151367 ms, max = 0.373779 ms, mean = 0.15368 ms, median = 0.153564 ms, percentile(99%) = 0.156677 ms
[03/07/2022-05:05:52] [I] D2H Latency: min = 0.00170898 ms, max = 0.0136719 ms, mean = 0.00237128 ms, median = 0.00219727 ms, percentile(99%) = 0.00341797 ms
[03/07/2022-05:05:52] [I] Total Host Walltime: 3.00036 s
[03/07/2022-05:05:52] [I] Total GPU Compute Time: 2.96525 s
[03/07/2022-05:05:52] [I] Explanations of the performance metrics are prin
[03/07/2022-05:05:52] [V]
[03/07/2022-05:05:52] [V] === Explanations of the performance metrics ===
[03/07/2022-05:05:52] [V] Total Host Walltime: the host walltime from when the first query (after warmups) is enqueued to when the last query is completed.
[03/07/2022-05:05:52] [V] GPU Compute Time: the GPU latency to execute the kernels for a query.
[03/07/2022-05:05:52] [V] Total GPU Compute Time: the summation of the GPU Compute Time of all
[03/07/2022-05:05:52] [V] Throughput: the observed throughput computed by dividing the number o
[03/07/2022-05:05:52] [V] Enqueue Time: the host latency to enqueue a query. If this is longer
[03/07/2022-05:05:52] [V] H2D Latency: the latency for host-to-device data transfers for input
[03/07/2022-05:05:52] [V] D2H Latency: the latency for device-to-host data transfers for output
[03/07/2022-05:05:52] [V] Latency: the summation of H2D Latency, GPU Compute Time, and D2H Later
[03/07/2022-05:05:52] [V] End-to-End Host Latency: the duration from when the H2D of a query is
[03/07/2022-05:05:52] [I]
&&&& PASSED TensorRT.trtexec [TensorRT v8003] # trtexec --onnx=model.onnx --minShapes=x:0:1x1x28
[03/07/2022-05:05:52] [I] [TRT] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +0, now: CPU
```



TRT如何测试并调优

nvvp



TRT如何测试并调优

Nsight Systems，最大区别，能记录CPU活动

CUDA+TensorRT调试建议

报错的地方并一定是出错的地方

基础

1. 二分注释；
2. kernel中，在特定线程id下打印输出值；
3. 函数返回结果检查；
4. cuda-memcheck + trtexec。

进阶

1. 写CUDA代码之前多想，甚至可以先写伪代码；
2. 多看开源代码，学习别人的写法，锻炼并行思维；
3. 多写单元测试代码，善用pytorch和libtorch可以节省不少时间；
4. 练习肉眼看代码，在大脑里运行代码。

https://blog.csdn.net/litdaguang/article/details/50462325

AIer 如何快速上手CUDA

如何快速找到需要的CUDA kernel代码?

PyTorch github layernorm 性能不够好

开源推理库的源代码，比如TNN，MNN等

多看开源代码，进行积累

如何快速找到需要的plugin代码?

TensorRT github

FastTransformer

Plugin自动生成工具

Libtorch 快速构建plugin

缺点：1、依赖libtorch；2、实现里有lower，速度不够好

AIer 如何快速上手CUDA

并行思维锻炼

```python
# 有以下python代码，使用cuda进行加速
import numpy as np
A = np.random.randn(200, 250)
B = np.random.randn(155, 200)
C = np.random.randn(155, 200)
D = A[10:165, 20:220] * B
C = C / C.max()
E = D * C
```

第一层：5个kernel

(1) $A = A[10:165, 20:220]$

(2) $D = A * B$

(3) $C\_max = C.\max(\ \ )$

(4) $C = C/C\_max$

(5) $E = D * C$

AIer 如何快速上手CUDA

并行思维锻炼

```python
# 有以下python代码，使用cuda进行加速
import numpy as np
A = np.random.randn(200, 250)
B = np.random.randn(155, 200)
C = np.random.randn(155, 200)
D = A[10:165, 20:220] * B
C = C / C.max()
E = D * C
```

第一层：5个kernel

第二层：3个kernel

(1) $D = A[10:165, 20:220] * B$

(2) $C\_max = C.\max(\ )$

(3) $E = D * C/C\_max$

AIer 如何快速上手CUDA

并行思维锻炼

```python
# 有以下python代码，使用cuda进行加速
import numpy as np
A = np.random.randn(200, 250)
B = np.random.randn(155, 200)
C = np.random.randn(155, 200)
D = A[10:165, 20:220] * B
C = C / C.max()
E = D * C
```

第一层：5个kernel

第二层：3个kernel

第三层：stream or 1个kernel

Stream版

Stream0: $D = A[10:165, 20:220] * B$

Stream1: $C\_max = C.\max(\ \ )$

Stream0: $E = D * C/C\_max$

还可以合并成一个kernel

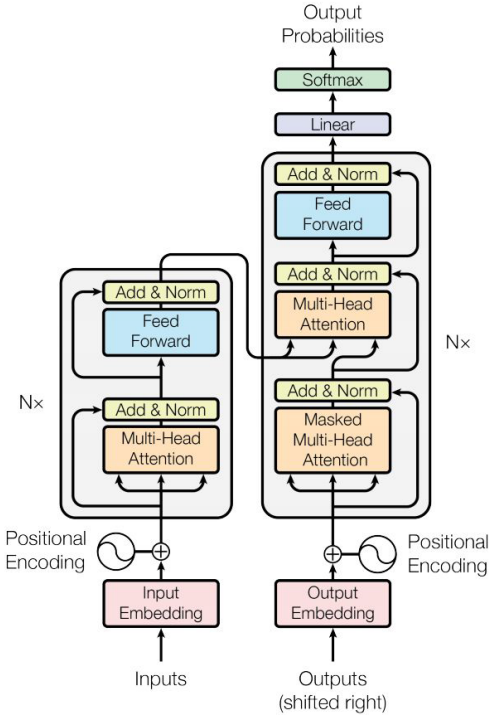
模型可以加速到什么程度？以Transformer模型为例

BERT和ViT都属于Transformer模型，结构很像，其核心是attention层。

下面以BERT模型举例。需要对BERT模型结构非常了解。

开始：拿到ONNX模型，使用API的方式编写构建代码，生成FP32 base模型。

第一步：进行FP16实现。修改小，收益大，但有可能溢出导致精度损失；



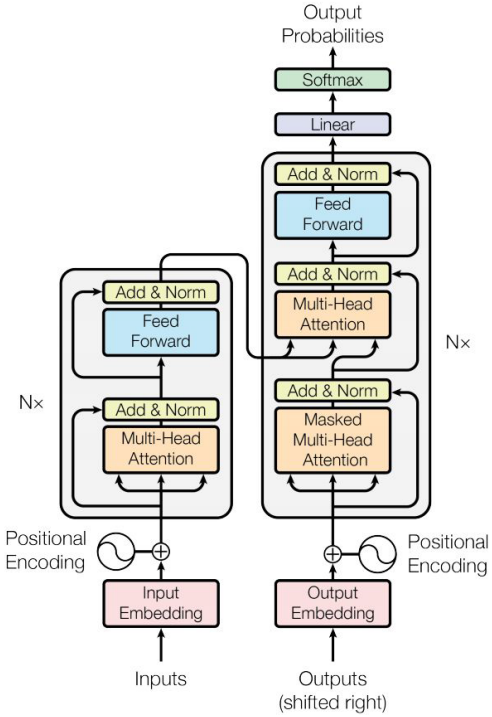
模型可以加速到什么程度？以Transformer模型为例

BERT和ViT都属于Transformer模型，结构很像，其核心是attention层。下面以BERT模型举例。需要对BERT模型结构非常了解。

开始：拿到ONNX模型，使用API的方式编写构建代码，生成FP32 base模型。

第一步：进行FP16实现。修改小，收益大，但有可能溢出导致精度损失；

第二步：进行算子合并。有三个模块可以合并，分别为embLayernorm、QKV2CTX，SkipLayernorm。需要对QKV2CTX之前的QKV矩阵乘做合并。修改大，收益大。



模型可以加速到什么程度？以Transformer模型为例

第四步：做INT8，准备数据，编写calibrator代码。

修改不大，收益一般，更容易溢出。

经验上，比较好的显卡，T4及以上，跟fp32比，fp16和int8的收益差不多，fp16+int8获得最大收益。

第五步：varlen，即拼batch时不padding。修改大，收益大。

模型可以加速到什么程度？以Transformer模型为例

第四步：做INT8，准备数据，编写calibrator代码。

修改不大，收益一般，更容易溢出。

经验上，比较好的显卡，fp16和int8的收益差不多，fp16+int8获得最大收益。

第五步：varlen，即拼batch时不padding。修改大，收益大。

第六步：合并算子第二轮加速。修改不大，收益一般。

（1）通过修改QKV合并矩阵的权值，可以省掉QKV2CTX中的两次转置。

（2）layernorm可以将两次归为一次。

模型可以加速到什么程度？以Transformer模型为例

第七步：合并算子第三轮加速。修改很大，收益大。目前只有NV的人能做。

（1）针对与特殊输入维度加速，合并成一个kernel，比如dim=64,96,128；

（2）除了对fc和conv层做int8，对QKV2CTX做int8。

第八步：做稀疏化矩阵乘，需要模型训练配合。

第九步：做INT4，需要模型训练配合，只能在超大plugin方式内做。

第十步：做cuda-graph，在超大plugin方式内做更方便。

深蓝学院
shenlanxueyuan.com

感谢聆听！

Thanks for Listening