cuda与tensorrt模型部署一百问！

汽车人 自动驾驶之心 2023-10-12 07:30 发表于广东

论文作者 | 汽车人
编辑 | 自动驾驶之心

Q1: GPU版本，CUDA版本与TensorRT版本之间的关系

答：这三者的版本有对应关系，具体来说：高版本的GPU驱动向下兼容CUDA版本；下载TensorRT前需要确认cuda版本。具体参考以下链接：

https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

GitHub - NVIDIA/TensorRT: NVIDIA® TensorRT™, an SDK for high-performance deep learning inference, includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for inference applications.

Q2: 代码里的 grid/block 对应硬件上的 SM 的关系是什么

回答：首先需要理解grid/block是软件层的概念，而SM是硬件层的概念。所以我们在GPU中是找不到grid/block的，所以只能抽象去理解这个关系。一般来讲一个kernel对应一个grid，分给多个SM去处理。之后每一个SM去处理一个grid中的多个block。这里需要注意的是，block不可以跨越SM去分配，也就是一个block里面的多线程统一由同一个SM中分配资源。因为block中的thread是共享资源的(比如shared memory)。

Q3: 各位佬，我有一个问题，jetson系列，一般都是共享内存，是不是不需要使用cudaMemcpy这个函数了? 要使用其他的memcpy方式吗?

关于共享内存在英伟达官方做了一个简短的介绍，链接如下，帮助理解

在 CUDA C / C ++ 中使用共享内存 - NVIDIA 技术博客

对于共享内存的shared-memory-cuda-cc/使用，Jetson系列确实可以直接访问共享内存而无需使用cudaMemcpy函数。首先，理解一下cudaMemcpy函数的功能: (库函数官方介绍)

NVIDIA CUDA Library: cudaMemcpy (horacio9573.no-ip.org)

从这个函数的介绍，翻译理解一下是将 count 个字节从 src 指向的内存区域复制到 dst 指向的内存区域。是将一个内存空间中的数据复制到另个内存空间中。关于这个函数及相关函数的用法，主要是用于主机内存与GPU内存之间的数据传输，或者是其他内存间的拷贝工作。而共享内存用于 同一个线程块内的线程之间共享数据，所以不涉及到内存数据的转移的话，不用copy函数。故 得出上述结论。。

这里提问者估计混淆了一个概念，你这里想表达的是统一内存(unified memory)而不是共享内存(shared memory)。shared memory无论是不是jetson，只要是GPU一般都会有的概念。而unified memory是Jetson中的概念，表示的是CPU和GPU共享同一片“虚拟”内存(注意这里实际意义上还不是共享同一片物理内存)。所以也就没有了CPU到GPU的数据拷贝过程。使用unified memory的编程方式跟平时有一些差异，你可以看看这篇文章，写的比较详细。以及官方文档

https://developer.ridgerun.com/wiki/index.php?title=NVIDIA_CUDA_Memory_Management#Unified_Memory_Programming_.28UM.29

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#unified-memory-programming

本文均出自《全搞定！基于TensorRT的CNN/Transformer/检测/BEV模型四大部署代码+CUDA加速！》

Q4: host内存应该不能直接传到share memory吧？肯定要过一次显存，我理解的没问题吧？ 如果遇到只需要读一次的情况，比如说resize操作，是不是就不需要用到共享内存了呢

回答：嗯，shared memory中的数据是从显存(global memory)中取出来的，所以需要先过一次显存。默认下kernel中如果没有特殊指定，会跳过shared memory直接从global memory中

取数据。所以你说的只读一次的情况是可以不用共享内存的。

Q5: 为什么trtexec转换engine时，采用FP16推理、INT8量化，推理延时可能变得更久？

可能原因是：

a. 量化后可能会引入一些多余的计算操作和内部的一些reshape。对于小模型，多余的计算带来的延时并不明显；而reshape会涉及一些内存操作，这个是延时变长的主要原因。对于reshape引起的延时变长，我们的解决办法是让TensorRT不做一些额外的这些操作，但TensorRT内部产生的reshape我们没有办法解决的。

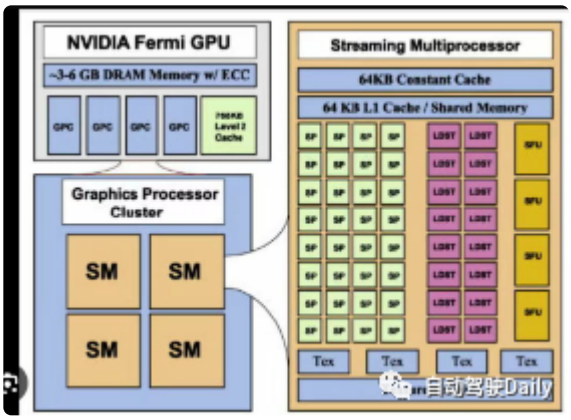
b. 另外，TensorRT有kernel auto tuning的机制，因此选择的kernel不一定是效率最高的。

Q6: 什么是Myelin？

这是TensorRT内部的一个概念，负责graph compilation（图编译）和execution backend（执行后端）的内容。

Q7: constant cache和constant memory的区别？

constant cache和constant memory是两个概念，cache更靠近计算单元，所以速度更快。constant cache是以前GPU版本中的概念，比如早期Fermi架构的SM block（上图）。而现在Ampere架构的SM如下图所示。



Q8: 在cuda, cudnn, tensorrt版本相同的情况下，可以将其他电脑上转换好的trt直接在自己电脑运行吗？

不同的GPU架构针对trt的优化方式不一样，所以移植到另外一个平台可能会不兼容。

Q9: 模型部署后，用什么手段分析推理性能？

可以利用Nsight工具分析模型推理性能。通过该工具可以捕获模型各个kernel运行的时间。针对运行情况，我们再做优化。

Q10:神经网络中吞吐和延迟的关系？

吞吐是用来描述一个硬件设备单位时间内可以完成的计算量；延迟是用来描述一个模型推理所需的时间。延迟又分为计算产生的延迟和数据传输（包括数据同步）造成的延迟。我们可以用nsys和Nsight Compute工具定量分析不同阶段的延时情况。

Q11: tensorrt量化方法？

trt默认和推荐的量化算法是entropy，但具体需要看情况，有时候选择minmax或者percentile会达到更好的效果。这个需要结合op的特点一起考虑。

Q12: 模型导出fp32的trt engine没有明显精度损失，导出fp16损失很明显，可能的原因有哪些？

比较突出的几个可能性就是：对一些敏感层进行了量化导致掉精度比较严重，或者权重的分布没有集中导致量化的dynamic range的选择让很多有数值的权重都归0了。另外，

minmax, entropy, percentile这些计算scale的选择没有根据op进行针对性的选择也会出现掉点。

Q13: onnx模型推理结果正确，但tensorRT量化后的推理结果不正确，大概原因有哪些？

出现这种问题的时候，需要先确认两种模型推理的前处理（例如，对输入的各种预处理需要和pytoch模型的训练预处理完全一致）和后处理是否一致。确认是量化引起的问题时，可能原因有：

a. calibrator的算法选择不对；

b. calibration过程使用的数据不够；

c. 对网络敏感层进行了量化；

d. 对某些算子选择了不适合OP特性的scale计算。

Q14: 采用tensorRT PTQ量化时，若用不同batchsize校正出来模型精度不一致，这个现象是否正常？

这个现象是正常的，因为calibration（校正）是以tensor为单位计算的。对于每次计算，如果histogram的最大值需要更新，那么PTQ会把histogram的range进行翻倍。

参考链接：https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#enable_int8_c

不考虑内存不足的问题，推荐使用更大的batch_size，这样每个batch中包含样本更加丰富，校准后的精度会更好。但具体设置多大，需要通过实验确定（从大的batch size开始测试。一点一点往下减）。需要注意的是batch_size越大，校准时间越长。

Q15: 关于对齐内存访问的疑问：如果使用L1cache，访问的颗粒度为128B，对齐的首地址应该为128B偶数倍，不应该是0B，256B，512B.....吗？

实际上这里的偶数倍（even multiple）指的是地址是偶数倍的，并非128B的偶数倍。比较官方的解释可以参考如下链接：

https://www.nvidia.com/content/PDF/sc_2010/CUDA_Tutorial/SC10_Fundamental_Optimizations.pdf（P.8后内容中有介绍）

Q16: 如何使用**nsight或CUDA runtime api分析模型推理性能？**

通过nsight可以看到核函数的名字（可通过名字推测它是用cuda core或tensor core, fp16还是int8）还有可以查看memory的流动。

Q17: 如何尽量减少GPU和CPU之间的数据交互或内存分配与回收**？**

由于在推理过程中，CPU与GPU之间的数据拷贝耗时较长或出现频繁分配和回收内存的现象，这大大降低了模型推理性能。我们可以采用在推理模型前分配好所需要的最大内存（做到内存复用）以降低内存分配或回收的次数。针对CPU与GPU之间数据相互拷贝问题，我们需要优化代码流程，尽量减少拷贝的次数或寻找更好的方法去掩盖这个动作需要的时间。

Q18: 如果QAT可以使模型尽可能减少量化带来的误差，那么可以不做敏感层分析，直接将整个网络量化为INT8吗？

不建议这么做，从经验来看，敏感层量化到INT8精度会下降很多，所以还是有必要进行敏感层分析。

Q19: 模型量化到INT8后，推理时间反而比FP16慢，这正常吗？

正常的，这可能是tensorrt中内核auto tuning机制作怪（会把所有的优化策略都运行一遍，结果发现量化后涉及一堆其他的操作，反而效率不高，索性使用cuda core，而非tensorrt core）。当网络参数和模型架构设计不合理时，trt会添加额外的处理，导致INT8推理时间比FP16长。我们可以通过trt-engine explorer工具可视化engine模型看到。

Q20: 请教一下，engine推理的时候，batchsize=1和batchsize=4，推理时间相差也接近4倍合理吗？有什么办法让多batch的推理时间接近单batch吗？比如加大显存？

回答：(韩君)这个可能出现的原因有很多，有可能单个batchsize的推理就已经把GPU资源全部吃满了，所以batchsize=4的时候看似加大了并行度，实际上也可能是在串行。建议把模型推理放在nsight system上分析一下，看看硬件资源占用率。

Q21: 在device固定的情况下呢？有什么参数设置或者增加streams的方式吗？试过把workspace设到最大，只有轻微的提升

回答：(韩君)workspace的大小跟性能提升关联不大，workspace是使用在创建推理引擎时TensorRT选择tactics来进行优化的，workspace越大可以选择的tactics越丰富。但除非特别的小，一般关联不是那么大。试试fp16, int8这种量化参数来试试量化，cuda-graph来试试kernel launch的隐藏，builderOptimizationLevel的等级设置高一点等等。光靠参数优化还是有点局限。可以看看模型架构是否有冗长。