PyConChina 2019

Python机器学习性能优化

以BERT服务为例，从1到1000

刘欣

目录 CONTENTS

1. 优化的哲学

"There ain't no such thing as a free lunch"

Ahmdal's Law

- 系统整体的优化，取决于热点部分的占比和该部分的加速程度

$$Speedup = \frac{time_{old}}{time_{new}} = \frac{1}{(1-func_{cost})+func_{cost}/func_{speedup}}$$

$$Speedup_{func_{speedup}=\infty} = \frac{1}{1-func_{cost}}$$

No Free Lunch

- 定位热点 & 热点加速
- 对于项目开发周期：
  1. 先做出效果
  2. 确定整体pipeline
  3. 再考虑优化

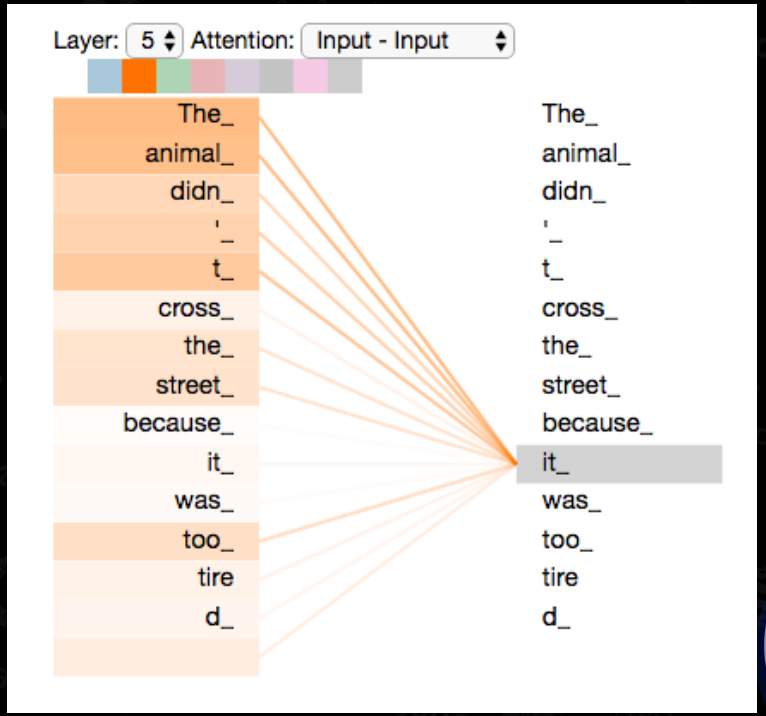
- 对于人工智能项目：迭代周期更长，更是如此

以BERT服务为例

- BERT：
  TODO： 一句话解释
- 横扫多项NLP任务的SOTA榜
- 惊人的3亿参数

以BERT服务为例

- Self Attention机制
- 预训练 + Finetune



以BERT服务为例

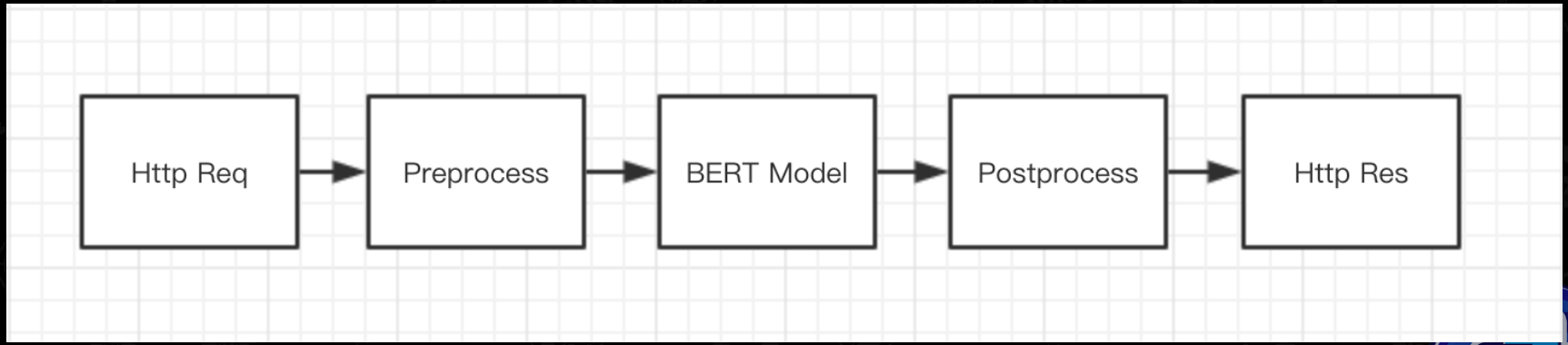
- 完型填空任务：
  Happy birthday to [MASK] .

- Web API:
  $ curl -X POST http://localhost:5005/predict -d 's=Happy birthday to [MASK].'

  [“you"]

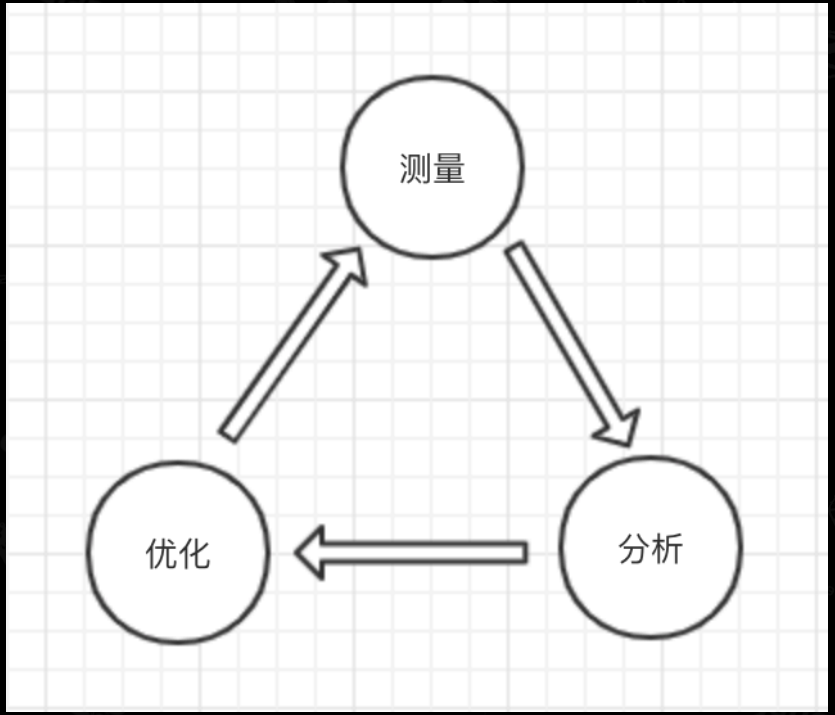
以BERT服务为例

- 我们现在上线了这样一个服务，每秒钟只能处理10个请求
- Q: 大家一开始如何着手优化



Profile before Optimizing

- 建立闭环



2 了解你的资源

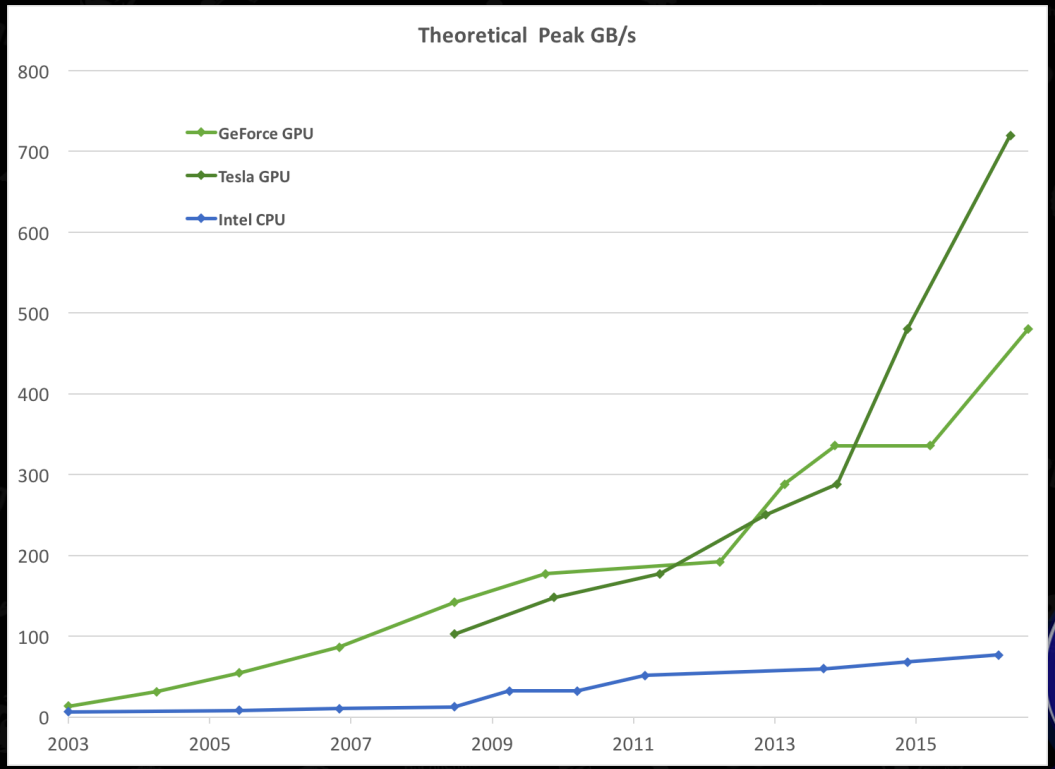
cpu/内存/io/gpu

GPU为什么“快”?

计算力对比

- GFLOPS/s

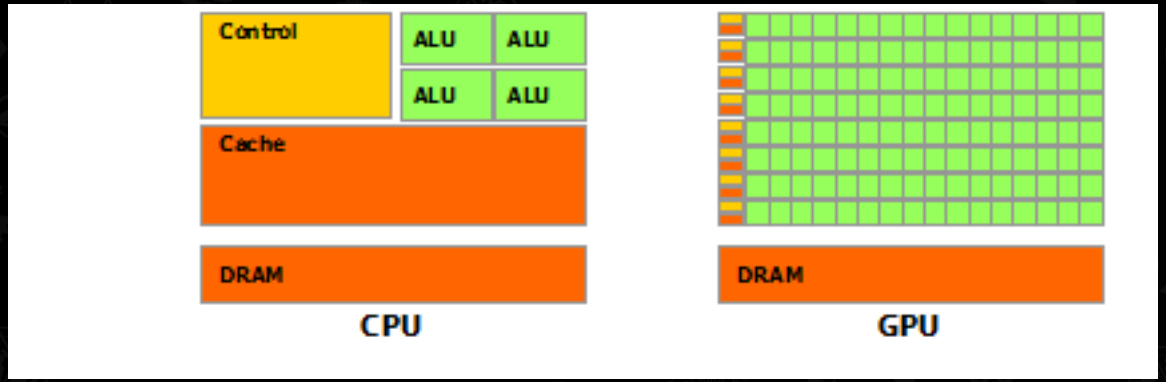
  每秒浮点数计算次数



摩尔定律的限制

- “集成电路上可容纳的晶体管数目，约每十八个月便会增加一倍”

CPU更多用在了Cache(L1/L2/L3)和Control
GPU绝大部分用来在了ALU计算单元



GPU特性

- SIMD
- 显存分级
- 异构&异步

Python为什么“慢”?

Flask Development Server

- 默认threaded server
- GIL限制多核使用
- 解释执行：序列化慢（动态特性的tradeoff）

Flask Production Server

- gunicorn 多进程解决多核利用率问题
- gevent 协程替代多线程网络模型
- 更高效的序列化lib

3 定位性能瓶颈

Profile before Optimizing

Python Profilers

- time.time()
- cProfile
- line profiler
- pyflame

line profiler

- 放个截图

cProfile

- 倒序打印 & graph

pyflame

- 插桩 or 采样
- 放个flamegraph
- 开源地址

wrk

- 制造压力
- 挖掘整体性能瓶颈
- 实现非常精妙的压力工具，强烈安利（要不要写个py binding）

4 动手优化

多线程服务器的问题

- 每个请求单独进GPU，利用率不高
- 大量请求并行，CUDA会爆
- wrk截图

service-streamer

- 请求排队组装成batch，再一起送进GPU
- 一个GPU worker只会有一条队列，最大batch size可控
- 多个GPU worker分布式处理
- todo：补图

batch predict profile

- 有了service-streamer：
  网络服务性能 等价与 本地batch predict的性能
- 再次profile：这里先卖个关子，猜猜哪一步是瓶颈
- Bert Tokenize远高于inference时间
- 再次说明：先profile再优化

pybind c++ extenstion

- pybind11
- 感谢知乎cuBERT提供的c++实现
- 用pybind11一波封装
- 再加上正经多线程

model inference optimize

- 终于到了我们直觉的优化部分
- 先补了补GPU和Cuda的知识
- 几个可以选择的方案：
  1. 买更多更贵的机器——fp16、v100、cpu化
  2. 优化算法——知识蒸馏
  3. 优化实现——jit/TensorRT

PyTorch jit

- 原理介绍
- 转化为graph截图

TensorRT

- NVIDIA推出的inference引擎
- 自家硬件使用到极致
- 与CPU比较：20x faster
- 正确的问法：
  与TF/PyTorch比较如何?

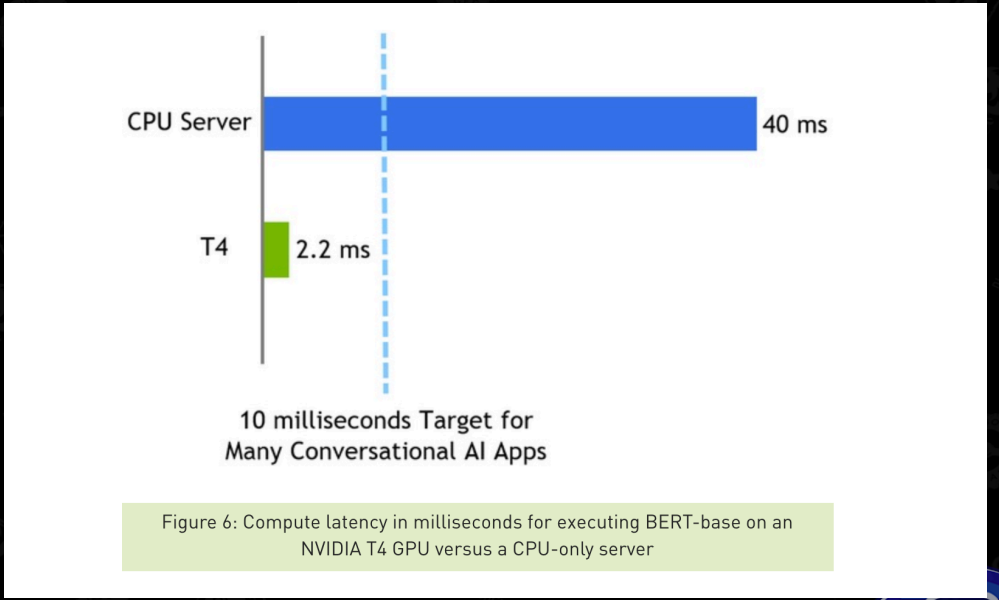


Figure 6: Compute latency in milliseconds for executing BERT-base on an NVIDIA T4 GPU versus a CPU-only server

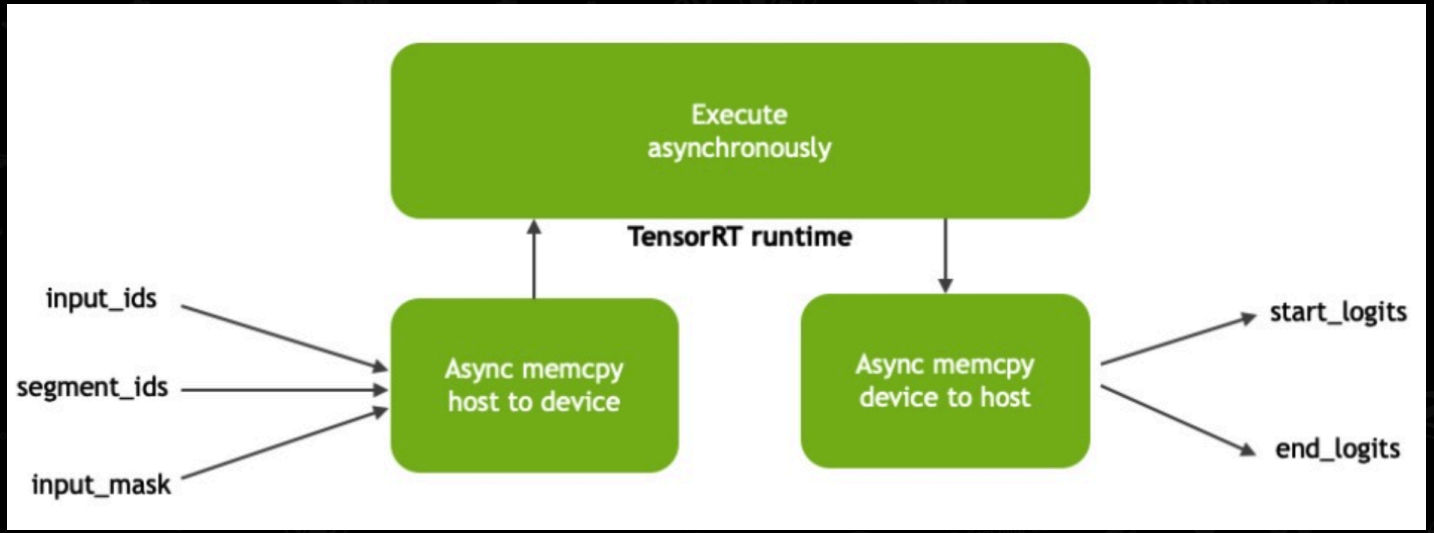
BERT runtime

- 使用SQuAD任务测试，输入padding到328，batch size分别取1和32
- 计时代码只包含GPU时间，排除掉前后处理时间，另包含数据在CPU和GPU之间copy的时间

bs * seqlen	tensorrt c++	tensorrt py	tensorflow	pytorch	pytorch jit
1 * 328	9.9	9.9	17	16.3	14.8
32 * 328	7.3		11.6	9.9	8.6

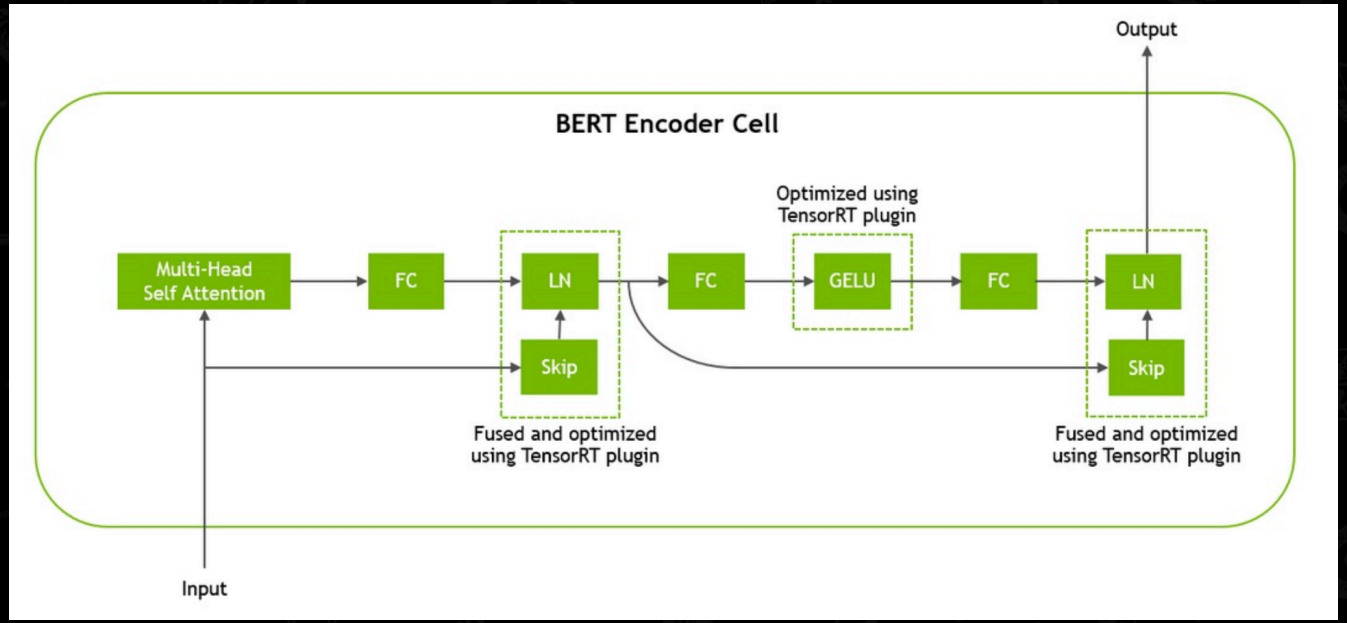
异步执行

- CPU与GPU异构，所以可以异步
- PyTorch也是异步执行，所以没有带来提升



cuda优化

- 更高效的kernel函数实现，替代默认导出的算子



知识蒸馏

- Teacher Student 学的更快
- Huggingface Distill BERT
- 12层 蒸出 6层

what's next?

- TensorRT inference server
  改变pipeline
- cpu化
  不在意延时，只追求吞吐量
- fp16低精度

THANK YOU

Meteorix 刘欣

github.com/Meteorix

15927607981