ERNIE BERT 分析

现状

基于百度GaoWei的分析, 最新的paddle主线， 20个线程的latency为60ms， 1个线程的latency为348.37ms

baseline：
在CPU上面的测试数据
@徐屹， 82.7681ms
intel BRTR 28.62ms

优化迭代：
82.7681 ms → 60.3766 ms (提升27%)

Paddle速度：

数据量	date	num threads	speed	top
10个数据	8-21	1	348.37	100%
10个数据	8-21	10	75.131	1000%
10个数据	8-21	20	60.3766	2000%
10个数据	8-21	40	88.5045	4000%

TF速度：

OMP_NUM_THREADS (env.sh)	TOP(%)	Runtime(ms)
1	102	289.050445
20	1934	27.697877
40	3665	32.155056

而我在Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz上（和6148配置相似）上复现出的结果如下
Paddle速度：

repeat=100

数据量	date	num threads	speed(ms)
10个数据	09-02	1	230.922
10个数据	09-02	20	55.745

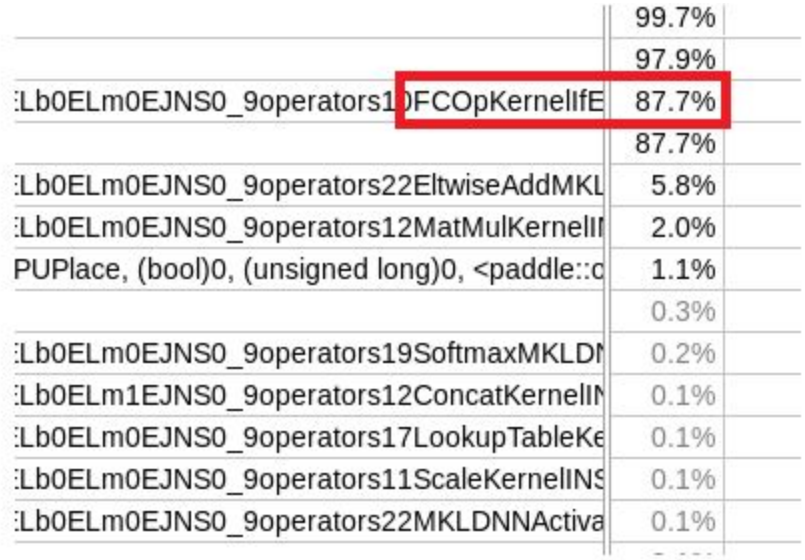
TF 速度：

OMP_NUM_THREADS	speed(ms)
1	210.182
20	27.588

1. **单线程上的优化**
    a. 将冗余的reshape和transpose去掉， PR #19585
    有一定的性能提升，但是提升空间有限（多线程也是），因为原本的transpose2和reshape2在BERT Base模型中的profile占比大概在2%左右

	Before	After	Gain
Latency	230.922	226.708	1.82%

    b. Vtune
    在加入上述优化后看Vtune的结果，我们可以清楚的看到接近90%的时间是运行在FC MKL的GEMM的实现中（已是目前最优的计算）， 其次elementwise_add也是用mkldnn优化过的。所以在单线程的情况下，在不考虑INT8优化的前提下，我们认为模型优化的已经较优。

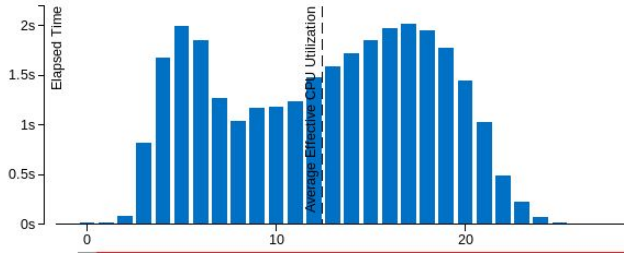


    c. 与竞品的差距
    竞品在单线程上做的优化行为是将图中所有的op fuse成一个大op，也就是可以节省一些framework的代价，但是这种优化不通用。且现在paddle和竞品在单线程上的差距10%左右（基于我复现的情况），属于合理范围。
    d. 下一步工作
    单线程的性能优化下一步可以考虑使用FC的INT8量化。基于现在的分析，FC的运算占到接近90%，所以INT8化FC会得到进一步的幅度较大的性能提升

2. **多线程上的优化**
    之前的优化的工作基本上都是基于单thread的优化，多线程的优化工作并未太多开展，所以会有很多优化空间。与竞品在多线程上的2倍多的差距初步怀疑是由于多线程的某个bug引起的，而非一些特殊的优化手法。

a. 利用Vtune的“Threading”模式分析

- 竞品



Spin and Overhead Time: **110.609s (22.9% of CPU Time)**

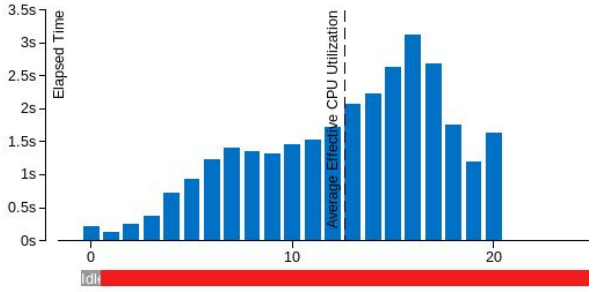
Top Functions with Spin or Overhead Time

The section lists top functions in your application with the most spin and overhead time.

Function	Module	Spin and Overhead Time	(% from CPU Time)
__kmp_fork_barrier	libiomp5.so	99.201s	20.5%
__kmp_join_barrier	libiomp5.so	6.005s	1.2%
__kmp_join_call	libiomp5.so	1.626s	0.3%
__kmp_api_GOMP_parallel_end	libiomp5.so	1.057s	0.2%
__kmp_fork_call	libiomp5.so	0.894s	0.2%
[Others]		1.826s	0.4%

*N/A is applied to non-summable metrics.

- Paddle



Spin and Overhead Time: **220.514s (36.8% of CPU Time)**

Top Functions with Spin or Overhead Time

The section lists top functions in your application with the most spin and overhead time.

Function	Module	Spin and Overhead Time	(% from CPU Time)
__kmp_fork_barrier	libiomp5.so	198.415s	33.1%
__kmp_join_barrier	libiomp5.so	12.149s	2.0%
__kmp_join_call	libiomp5.so	6.240s	1.0%
__kmp_GOMP_microtask_wrapper	libiomp5.so	2.080s	0.3%
__kmp_launch_thread	libiomp5.so	0.590s	0.1%
[Others]		1.040s	0.2%

*N/A is applied to non-summable metrics.

由上图比较可以看出， paddle多线程模式下有明显的锁的问题导致CPU的空转时间较长，paddle中各个线程的利用率也是有进一步的优化空间。具体导致的线程利用率不高的原因，需要进一步的分析vtune中的信息。

- **下一步工作**

1. 仔细查看paddle的“threading” vtune log，调查是多线程的锁是否正确释放
2. 总结竞品的所用的优化方法，探索是否可应用于paddle