

实验报告

硬件环境

Intel(R) Xeon(R) Silver 4215R CPU @ 3.20GHz
20 核 cpu

Github

<https://github.com/Nina-yang/maxpool-add>

代码结构

生成 test_case 的代码: generate_test_case.py

Tensor 定义: tensor.hpp

算子: max_pool_add.hpp

Fused 算子: fused_op.hpp

编译&运行 example: batch_test.sh

优化记录

version0: 使用多维 vector 实现

version1: 使用数组, 用维度映射压缩到单维

version2: 加入 omp 优化 for 循环; 调整 thread 数目和 static,dynamic 等参数。

1> thread 能被核数整除时优于不能整除时, 符合预期。Thread 过多会导致性能下降, 在该硬件上取 4 较优。运算量不同时, 通过 grid search 搜索最优的 thread 组合应该会更优, 但太耗时了没有仔细做。

version3: 加入 avx-512 指令集优化 add 运算

version4: v0 版先做 padding 再做 maxpool, 中间会发生一次数据拷贝, 较为耗时。该版本将 pad 和 maxpool 操作融合起来, 耗时能减少到原版的 1/4 左右。

version5: 将 boardcast 和 elementwise add 融合起来, 通过索引变换解决, 不去做数据拷贝。但是耗时反而增加了, sad。猜测可能是由于计算索引时加入了过多操作。

实验结果汇总

small 包含 100 个 case, mid 包含 100 个 case, large 包含 40 个 case, large_batch 包含 10 个 case

各测试集数据规模如下:

```
size_dict = {
    'small': { "C": [ 1, 4], "B": [ 1, 4], "H": [100, 200], "W": [100, 200] },
    'mid' : { "C": [ 4, 8], "B": [ 4, 8], "H": [200, 400], "W": [200, 400]},
    'large': { "C": [ 8, 16], "B": [ 8, 16], "H": [400, 800], "W": [400, 800]},
    'large_batch': { "C": [ 128, 256], "B": [ 32, 64], "H": [64, 65], "W": [64, 65]},
}
```

测试结果如下

millisecond	small	mid	large	large_batch
Baseline (B in brief)	2.8 \pm 0.6	569 \pm 5	4757 \pm 60	2322 \pm 87
B+omp	0.0 \pm 0.0	315 \pm 34	2065 \pm 136	1854 \pm 121
B+avx	2.9 \pm 1.0	553.5 \pm 2	4796 \pm 54	2314 \pm 79
B+omp+avx	0.1 \pm 0.3	341.0 \pm 18	2017 \pm 176	1854 \pm 121
B+omp+op fusion	-	-	-	523 \pm 61

参考资料

- [1] <https://zhuanlan.zhihu.com/p/518237328>
- [2] Intel CPU 支持指令集: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#techs=AVX,AVX_512&ig_expand=133,4330,4374,4330,4369,159&text=mm512_loadu_pd
- [3] openmp 入门教程: <https://zhuanlan.zhihu.com/p/397670985>
- [4] openmp 关键字: https://blog.csdn.net/qq_40765537/article/details/106025514
- [5] SIMD 指令介绍: <https://zhuanlan.zhihu.com/p/416172020>
- [6] SIMD 加速代码 example: <https://zhuanlan.zhihu.com/p/457505686>