第五届国产 CPU 并行应用挑战赛

初赛赛题

深度学习 Transformer 算子

赛题背景

自然语言技术近年来得到了长足的发展，AI 加速芯片的计算能力也飞速提升，自然语言领域出现了很多超大的模型，比如 BERT, GPT-2, GPT-3 等，在机器翻译、语言生成等方面得到了广泛的应用。这些模型，多是由 Transformer 模型架构发展而来。2017 年，Google 机器翻译团队发表了一篇名为《Attention is All You Need》的论文，提出了 Transformer 模型架构，不仅在预测结果上击败了当时其他的所有模型，而且由于其计算的可并行性大大加速了训练速度，受到了广泛的关注。

Transformer 架构的核心是 MultiHead-Attention 机制。MultiHead-Attention 机制涉及多个批量矩阵乘法操作，计算量比较大，需要针对实际使用的计算设备深入定制优化。因此，本次比赛希望开发并优化一个简化版本的 MultiHead-Attention 实现。

赛题简介

本赛题中 MultiHead-Attention 的算法如下:

1、 输入参数

MultiHead-Attention 的输入包含以下几个：

存储布局左高右低，比如[M, N]，表示 N 维度连续，M 为高维，N 为低维。

参数	输入/输出	说明	数据存储布局或类型
B	输入	Batch 大小	正整数 int
S	输入	Sequence，即序列长度	正整数 int
D	输入	词向量维度	正整数 int
N	输入	多头的头数	正整数 int
X	输入	输入编码过后的单词序列	[B, S, D] float
W	输入	Wq, Wk, Wv 三个权重连续排列	[3, D, D] float
Q,K,V,QK	临时存储空间	临时存储空间，可用来存储中间结果	Q,K,V 为[B,S,D] float, QK 为[B,N,S,S] float
Y	输出	输出的中间特征	[B, S, D] float

注意: W 的维度[3, D, D]，分别为[Wq, Wk, Wv]，Wq, Wk, Wv 的维度均为[D, D]

2、 算法介绍

注: * 代表矩阵乘法

1) 首先，使用输入的 X 和 W 的转置分别计算 Q, K, V (如图 1 中(a)过程)，此处 Wq,Wk,Wv 的最低维度是与 X 的公共维度，即相乘累加的维度:

$$\mathrm{Q}:[\mathrm{B},\mathrm{S},\mathrm{D}] = \mathrm{X}:[\mathrm{B},\mathrm{S},\mathrm{D}] * W_q^T:[\mathrm{D},\mathrm{D}]$$
$$\mathrm{K}:[\mathrm{B},\mathrm{S},\mathrm{D}] = \mathrm{X}:[\mathrm{B},\mathrm{S},\mathrm{D}] * W_k^T:[\mathrm{D},\mathrm{D}]$$
$$\mathrm{V}:[\mathrm{B},\mathrm{S},\mathrm{D}] = \mathrm{X}:[\mathrm{B},\mathrm{S},\mathrm{D}] * W_v^T:[\mathrm{D},\mathrm{D}]$$

2) 将 Q, K, V 转换为多头表示，为了简化，此处头的数目为 N, $\mathrm{D}' = \frac{D}{N}$ (如图 1 中(b)过程开始部分):

$$\mathrm{Q}:[\mathrm{B},\mathrm{S},\mathrm{D}] => Q':[B,N,S,D']$$
$$\mathrm{K}:[\mathrm{B},\mathrm{S},\mathrm{D}] => K':[B,N,S,D']$$
$$\mathrm{V}:[\mathrm{B},\mathrm{S},\mathrm{D}] => V':[B,N,S,D']$$

3) 分别将Q′的低两维乘以(矩阵乘)K′的低两维的转置,得到Q′K′ (如图 1 中(b)过程):

$$\mathrm{Q'K'}:[\mathrm{B},\mathrm{N},\mathrm{S},\mathrm{S}] = \mathrm{Q'}:[\mathrm{B},\mathrm{N},\mathrm{S},\mathrm{D}] * K'^T:[B,N,D,S]$$

4) 对Q′K′的**最低维度**执行归一化(原始操作为 softmax，此处为了方便比较精度，但是不影响计算逻辑，采用简化的归一化)操作，并除以$\sqrt{D'}$，得到Q′′K′′ (如图 1 中(c)过程):

$$\mathrm{Q''K''}:[\mathrm{B},\mathrm{N},\mathrm{S},\mathrm{S}] = \mathrm{Norm}(\frac{Q'K'}{\sqrt{D'}})$$

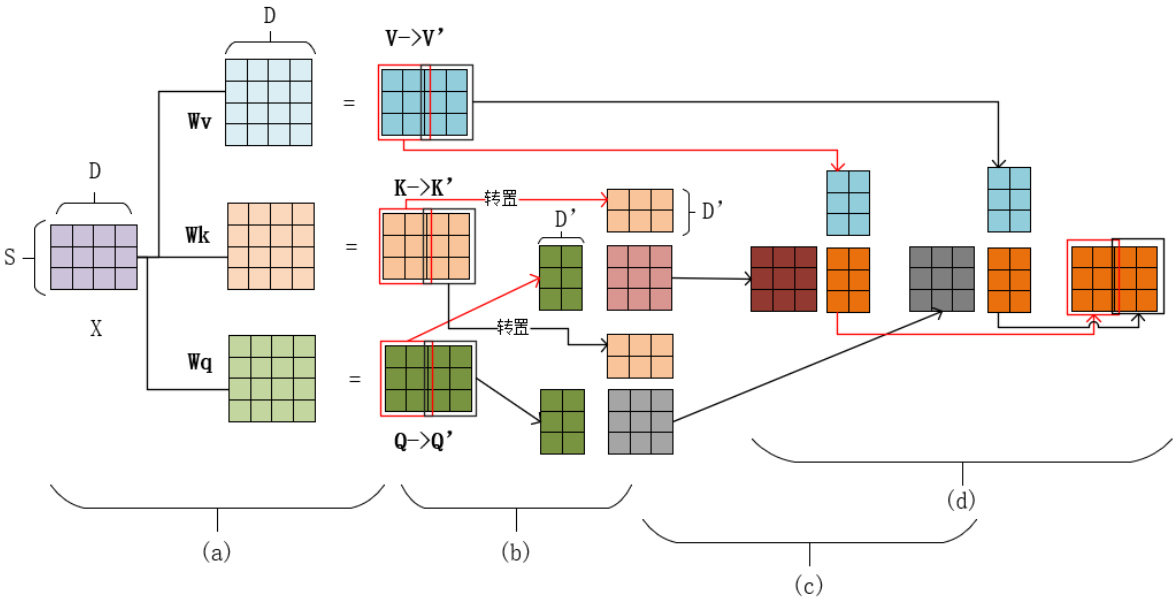
其中, Norm 的公式如下:

$$Norm(x_i) = \frac{x_i}{\sum_i x_i}$$

5) 分别将Q′′K′′的低两维乘以(矩阵乘) V′的低两维得到Q′′K′′V′ (如图 1 中(d)过程的前半部分)，并对结果的数据做转换，将头数 N 折叠回最后一维中(如图 1 中(d)过程后半部分)，得到最终的结果 Y：

$$\mathrm{Q''K''V'}:[\mathrm{B},\mathrm{N},\mathrm{S},\mathrm{D'}] = \mathrm{Q''K''}:[\mathrm{B},\mathrm{N},\mathrm{S},\mathrm{S}] * \mathrm{V'}[\mathrm{B},\mathrm{N},\mathrm{S},\mathrm{D'}]$$
$$Q''K''V':[B,N,S,D'] => Y:[B,S,D]$$

以 B=1, S=3, D=4, N=2 为例，下图展示了 MultiHead-Attention 的整个计算过程：



图表 1 MultiHead-Attention 的计算流程，其中 B=1，S=3，D=4，为了方便展示，此处头数为 2

3、 比赛参数描述

1) 输入参数范围:

a) B: [1, 128]

b) S: [1, 1024]，且大部分测例中 S 为 128 的倍数

c) D: 一般为 768, 1024, 1280, 1536 等数，使得 D/N 为 32 的倍数

d) N:一般为 12, 16, 24,32 等，使得 D/N 为 32 的倍数

2) 参数类型:

X, W: float*，每个 float 均在[0.0, 1.0]区间内

Q,K,V,QK: float*，为临时空间，在计算的时候，可以用于临时存储数据

B, S, D, N : int

Y:float*

代码结构

args.h	定义参数结构体	不可修改
util.h	辅助的函数以及宏定义，包括 LOG、浮点比较、对齐内存分配等函数	不可修改
function.h	功能函数接口	不可修改
function.c	功能函数接口实现，以及 naïve 的卷积实现	不可修改
master.c	主核函数	可以修改
slave.c	从核函数	可以修改
main.c	主函数	不可修改
data/	测试数据存放目录	不可修改
obj/	编译时的 obj 文件临时存储目录	不可修改

选手只能修改 slave.c 和 master.c 文件，不可对其他目录下文件进行更改。已有 Makefile 文件，可以直接 make clean && make 进行编译，make run, 即可自动执行测试，测试内容为 data/目录中的数据。

程序接口

本次比赛选手需要根据官方提供的数据结构及算子设计众核加速算法，直接接口如下图所示（master.c），multihead_attention 函数 athread_spawn 的方式调用了从核上执行 par_multihead_attn(slave.c)函数，执行计算，athread_join 则是等待所有的从核线程完成计算。**选手需要编写、优化 par_multihead_attn 函数，以及有需要的时候修改 multihead_attention 函数，其他文件均不可修改。**如果需要额外的主存空间，可以使用 util.h 中定义的 aligned_malloc/free 函数。

```
int multihead_attention(Args_t arg)
{
    athread_spawn(par_multihead_attn, arg); // spawn
    athread_join(); // wait for all slave threads finished
    return 0;
}
```

输入参数由官方提供，参数存储的结构体如下（args.h），如果还需要其他参数，则可以在 master.c 中再加入一层参数结构体定义。：

```
typedef struct Args
{
    int B; // batch
    int S; // sequence length
    int D; // vector size
    int N; // number of heads
    float* x; // input x
    float* w; // Wq, Wk, Wv
    float* Q;
    float* K;
    float* V;
    float* QK;
    float* y; // output
}Args, *Args_t;
```

为了方便理解，在 function.c 中提供了 naïve_multihead_attention 函数，为简单的串行实现，可以作为参考代码(如下图)，该代码对提供的简单测试样例正确通过。该实现代码仅供参考，具体计算说明以文档为准。

```
int run_multihead_attention(Args_t arg)
{
    //multihead_attention(arg);
    naive_multihead_attention(arg);
    return 0;
}
```

```
int naive_multihead_attention(Args_t arg)
{
    const int B = arg->B;
    const int S = arg->S;
    const int D = arg->D;
    const int N = arg->N;
    const float* x = arg->x;
    const float* w = arg->w;
    float* Q = arg->Q;
    float* K = arg->K;
    float* V = arg->V;
    float* QK = arg->QK;
    float* y = arg->y;
    const int PD = D/N;
    memset(Q, 0, sizeof(float)*B*S*D);
    memset(K, 0, sizeof(float)*B*S*D);
    memset(V, 0, sizeof(float)*B*S*D);
    memset(QK, 0, sizeof(float)*B*N*S*S);
    memset(y, 0, sizeof(float)*B*S*D);
    float* QN = (float*)aligned_malloc(sizeof(float)*B*N*S*PD, 128);
    float* KN = (float*)aligned_malloc(sizeof(float)*B*N*S*PD, 128);
    float* VN = (float*)aligned_malloc(sizeof(float)*B*N*S*PD, 128);
    //calculate Q, K, V
    for(int b = 0; b < B; b ++)
    {
        _local_gemm_rcr(x+b*S*D, D, w, D, Q+b*S*D, D, S, D, D);
        _local_gemm_rcr(x+b*S*D, D, w+D*D, D, K+b*S*D, D, S, D, D);
        _local_gemm_rcr(x+b*S*D, D, w+2*D*D, D, V+b*S*D, D, S, D, D);
    }
    _local_trans_head(Q, QN, B, S, D, N);
    _local_trans_head(K, KN, B, S, D, N);
    _local_trans_head(V, VN, B, S, D, N);
#define NI(b,n,s,pd) ((((b)*N+n)*S+s)*PD+pd)
#define QKI(b,n,sh,sl) ((((b)*N+n)*S+sh)*S+sl)
    // QK = Q*KT
    for(int b = 0; b < B; b ++)
        for(int n = 0; n < N; n ++)
            _local_gemm_rcr(QN+NI(b,n,0,0), PD, KN+NI(b,n,0,0), PD, QK+QKI(b,n,0,0), S, S, S, PD);

    double norm = sqrt(PD*1.0);
    for(int i = 0; i < B*N*S*S; i ++)
        QK[i] /= norm;
    for(int b = 0; b < B; b ++)
        for(int n = 0; n < N; n ++)
            for(int s = 0; s < S; s ++)
                _local_norm(QK+QKI(b,n,s,0), S);

    // reuse Q
    memset(QN, 0, sizeof(float)*B*S*D);
    for(int b = 0; b < B; b ++)
        for(int n = 0; n < N; n ++)
            _local_gemm_rrr(QK+QKI(b,n,0,0), S, VN+NI(b,n,0,0), PD, QN+NI(b,n,0,0), PD, S, PD, S);
    //trans back
    _local_trans_head_back(QN, y, B, S, D, N);

    aligned_free(QN);
    aligned_free(KN);
    aligned_free(VN);
    return 0;
}
```

data 目录下存储了 3 个简单的供测试的数据，其中*_arg 存储了参数内容，_data 存储了具体数据。选手不可更改此目录及其中的文件。程序会采用多次运行取平均值的方法，获得代码的运行时间，如下所示：

```
        MARK_TIME(start);
        // run your program
        for(int j = 0; j < REPEAT_N; j ++)
            run_convolution(arg);
        MARK_TIME(end);
        LOG("average time : %.3f ms", DIFF_TIME(start, end)/REPEAT_N);
```

赛事规则

1. 评分规则
   1) 比赛将使用组委会提供的一共十组测试样例进行测试，每组满分 10 分，共 100 分。评分所用的测试数据与题目中提供的测试数据均不相同；对于每组测试函数，程序将运行 20 次，统计从核接口函数的平均运行时间；对每组平均运行时间求和，按照总运行时间进行排名计算得分，运行最快者得 10 分。
   2) 选手必须保证计算结果的正确性，本次比赛采用绝对误差与相对误差对浮点数正确性进行判断，误差限为10^{-5}，任意一组测试函数计算结果错误，本次提交成绩无效。
   3) 选手成绩取个人多次提交里面的最好成绩。
   4) 如果时间一样，先提交的选手排名靠前。
   5) 选手优化可以从如何切分输入张量并尽可能重用数据的角度考虑，针对不同的参数范围，其最高效的划分方式可能不同。选手着重提高代码并行计算效率和发挥申威 CPU 计算潜力，参赛者若修改源码中已明确不能修改部分或做其他不合理的改动将导致成绩无效。
2. 编译选项
   编译链接选项位于 Makefile 文件中，不得更改。
3. 提交选项
   make run