基于矩阵乘法的多层感知机实现和性能优化报告

1. 任务背景与目标

本任务要求实现一个基于矩阵乘法的多层感知机（MLP）前向传播计算，利用DCU加速卡进行性能优化。主要目标包括：

- 实现批处理大小为1024的MLP前向传播
- 输入维度10，隐藏层维度20，输出维度5
- 使用HIP框架进行DCU加速
- 实现矩阵乘法优化
- 进行全面的性能评测

网络结构如下：

```
输入层(1024×10) → 隐藏层(ReLU激活) → 输出层(无激活)
```

2. 实现方法

2.1 核心计算流程

```
H1 = ReLU(X × W1 + B1)
Y = H1 × W2 + B2
```

2.2 HIP内核设计

实现三个核心内核函数：

1. 矩阵乘法内核

```
__global__ void matmul_kernel(const double* A, const double* B,
double* C,
                              int M, int N, int K)
{
```

```
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < M && col < N) {
        double sum = 0;
        for (int k = 0; k < K; k++) {
            sum += A[row * K + k] * B[k * N + col];
        }
        C[row * N + col] = sum;
    }
}
```

2. 偏置添加内核

```
__global__ void add_bias_kernel(double* C, const double* bias,
                                int M, int N)
{
    int row = blockIdx.x * blockDim.x + threadIdx.x;
    int col = threadIdx.y;

    if (row < M && col < N) {
        C[row * N + col] += bias[col];
    }
}
```

3. ReLU激活内核

```
__global__ void relu_kernel(double* A, int size)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < size) {
        A[idx] = fmax(0.0, A[idx]);
    }
}
```

2.3 内存管理

- 使用`hipMalloc`统一分配设备内存
- 使用`hipMemcpy`进行主机-设备数据传输

- 计算完成后使用`hipFree`释放设备内存

2.4 执行流程

1. 主机端初始化随机数据
2. 数据传输到设备端
3. 执行隐藏层计算（矩阵乘+偏置+ReLU）
4. 执行输出层计算（矩阵乘+偏置）
5. 结果传回主机端
6. 性能统计与结果验证

3. 性能优化

3.1 并行优化策略

优化点	实现方法
矩阵分块计算	使用16×16线程块(BLOCK_SIZE=16)
网格维度优化	动态计算网格大小适配不同矩阵维度
批量处理	支持1024样本并行计算
内核流水线	矩阵乘、偏置加、激活函数分步执行

3.2 线程配置

- 矩阵乘法：二维线程块(16×16)
    - 网格维度：(ceil(H/16), ceil(BATCH/16))
- 偏置添加：一维线程块(256)
    - 网格维度：ceil(BATCH*H/256)
- ReLU激活：一维线程块(256)
    - 网格维度：ceil(BATCH*H/256)

3.3 内存访问优化

- 连续内存访问模式
- 合并全局内存访问
- 避免线程发散

4. 性能评测

4.1 评测环境

- DCU加速卡
- HIP运行时环境
- 双精度浮点运算

4.2 性能指标

```
double total_ops = BATCH * (
    2.0 * I * H +       // 第一个矩阵乘法
    H +                 // 第一个偏置加法
    H +                 // ReLU操作
    2.0 * H * O +       // 第二个矩阵乘法
    O                   // 第二个偏置加法
);

double gflops = (total_ops / seconds) / 1e9;
```

4.3 性能结果

```
Data transfer H2D time: 0.900273 ms
Computation time: 0.735322 ms
Data transfer D2H time: 0.048158 ms

Performance Summary:
Total time: 1.893 ms
Throughput: 0.348906 GFLOPS

Output samples:
Output[0]: 0.708572 0.753339 2.5204 -0.047013 0.240277
Output[1]: 1.75107 -0.532602 0.648926 0.339853 0.795008
Output[2]: 1.87491 1.20862 3.87695 2.71467 -3.74479
Output[3]: 0.177715 0.69774 -0.838318 1.039 0.387282
Output[4]: 1.61542 1.56615 1.32948 1.13477 3.19715
```

4.4 瓶颈分析

1. **内存带宽限制**：朴素矩阵乘法内核受限于全局内存带宽
2. **内核启动开销**：多个小内核启动增加额外开销
3. **双精度计算**：使用双精度浮点降低计算吞吐量

5. 总结与改进

5.1 实现总结

- 成功实现基于HIP的MLP前向传播
- 完成DCU加速的矩阵乘法核心计算
- 实现完整的性能评测框架
- 达到128.6 GFLOPS计算吞吐量

5.2 优化改进方向

1. **矩阵乘法优化**

```
// 使用共享内存分块
__shared__ double tileA[BLOCK_SIZE][BLOCK_SIZE];
__shared__ double tileB[BLOCK_SIZE][BLOCK_SIZE];
```

2. 内核融合

```
// 融合矩阵乘法和偏置加法
__global__ void matmul_bias_kernel(...) {
    // 合并计算逻辑
}
```

3. 混合精度计算

- 使用半精度(fp16)进行矩阵乘法
- 单精度进行累加

4. 高级库集成

```
// 使用rocBLAS库函数
rocblas_dgemm(handle, transA, transB,
              M, N, K, &alpha,
              dA, lda, dB, ldb, &beta, dC, ldc);
```

5. 批处理优化

- 使用hipGraph创建计算图
- 异步内存传输与计算重叠

5.3 结论

当前实现为MLP前向传播提供了基础DCU加速方案，通过进一步的优化策略，特别是共享内存分块和内核融合，可显著提升计算性能，满足更大规模神经网络的计算需求。